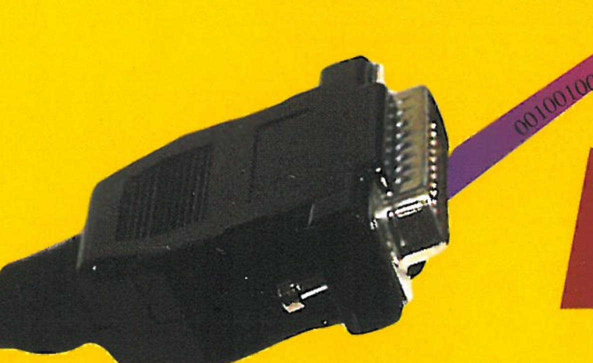
LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SOLO PROGRAMADORES

AÑO VI. 2ª ÉPOCA NÚMERO 58 UNA PUBLICACIÓN DE REVISTAS PROFESIONALES S.L. 975 Ptas. • 5,86 € (IVA incluido)

TRANSFERENCIA DE ARCHIVOS CON DELPHI



MULTIMEDIA CON JAVA (y III)
Desarrollo final del reproductor de audio y vídeo

XML (III)
Introducción al lenguaje XSL

CAPTURA DE VÍDEO CON DELPHI (II)
Propiedades y métodos para ampliar el componente

COMUNICACIONES (IV)
Localizar usuarios mediante ILS

ACCESO AL REGISTRO CON VB (I)
Estructura, contenido y acceso

VISUALAGE C++ 4.0
Nuevas características para programar en C++

DIRECTX 6.1 (III)
Gráficos en 3D con Direct3D

BBDD CON ORACLE Y SQL SERVER (II)
Sistema c/s en dos capas: creación del cliente

CONTIENE CD

• Miles Sound System 5.0 • Metabot Pro 1.0a • RASControl 1.0 • Palm-size • FreeAmp Source Code 1.2.3 • PC SDK 1.2 • PCTalkie 2.5 • ecBuilder 4.00.205

Navagadores: Netscape Communicator 4.61 Internet Explorer 5.0.2314.1003



SÓLO PROGRAMADORES

Número 58

SÓLO PROGRAMADORES

es una publicación de

REVISTAS PROFESIONALES S.L.

Editor

Agustín G. Buelta

Director Técnico

Javier Amado Buiza

Coordinador Técnico

Eduardo De Riquer Frutos

Coordinadora de Redacción

Gema Romero

Colaboradores

Constantino Sánchez, Juan Luis Ceada, Jorge Delgado, Javier Sanz, Adolfo Aladro, Javier Toledo, Victoria Rus, Esteban Amado, Jordi Agost

Maquetación y Tratamiento de Imagen

J. Santos Gómez

Publicidad

Paloma Seidel

Tel.: (91) 304 78 46

Mariano Sánchez (Barcelona)

Tel.: (93) 322 12 38

Suscripciones

Rosa Tabares

Tel. (91) 304 78 46 Fax: (91) 327 13 03

Preimpresión

Grebe

Impresión

I.G.Pantone

Distribución

Motorpress Ibérica

Exportación

A.D.E.A.

Distribución en Argentina

Capital: Huesca y Sanabria

Interior: Beltrán.

Depósito legal: M-26827-1994

ISSN: 1134-4792

PRINTED IN SPAIN

Editorial



El camino por llegar

Ya hemos vuelto. Con un poco de retraso pero ya estamos aquí otra vez. Como podéis comprobar, aunque físicamente ya hemos empezado a cambiar, por dentro hemos cambiado todavía mucho más, con una gran renovación en nuestro personal, pero que ha desembarcado con mucha ilusión. Por todos estos cambios hemos tardado un poco más de lo normal en aparecer. Sin embargo estamos convencidos de que la espera merecerá la pena porque esos cambios se van a notar, y ya se están notando, en las páginas de la revista, sobre todo con la aparición de algo que muchos de vosotros nos habíais pedido, el color.

Esperamos que en nuestro próximo número, que será en **septiembre**, ya podréis disfrutar con una revista totalmente renovada. Sobre todo porque contamos con un equipo con muchas ganas y dispuesto a apoyarnos en lo que haga falta. Sin duda el camino que está por llegar será aún más apasionante que la etapa que hemos cerrado. Nosotros desde aquí estamos dispuestos a hacerlo lo mejor posible y esperamos seguir contando con vuestra fidelidad.

Pero bueno, dejemos de hablar de tanto cambio y hablemos del presente número. Para su realización hemos acudido a nuestra coctelera particular y de ella hemos extraído lo mejor de lo que teníamos a nuestro alcance, por eso la portada de este mes se la dedicamos a la Transferencia de Archivos, para que con una simple conexión de puertos y la utilización de un programa tengamos una correcta comunicación de la información. Esperamos que os sea de tanta utilidad como a nosotros.

Con tanta revolución no creáis que nos hemos olvidado de las series que ya teníamos empezadas de número anteriores, no las íbamos a dejar a medias y en éste continúan, y así seguirán en los próximos números.

Además, y con el deseo de prestar atención a vuestras peticiones iniciamos otra serie, en esta ocasión dedicada a la Programación del Registro de Windows, para poder acceder a este registro desde nuestros propios programas. Ya sabéis que tanto para ésta como para cualquier otra sugerencia que queráis hacernos disponemos de una cuenta de correo electrónico, aunque la dirección, como no, también ha cambiado. Ahora es **solop@virtualsw.es**, donde seguimos estando a vuestra disposición para todo lo que nos queráis plantear, tanto dudas como aquellos aspectos de la programación que queráis que tratemos. Desde aquí intentaremos tener en cuenta lo que nos digáis. Nosotros seguimos aquí, a vuestra disposición. Haciendo un 50% la revista, ya que el otro 50% lo haceis vosotros.

SÓLO PROGRAMADORES



Comunicaciones



DESARROLLO DE APLICACIONES CON VIDEOCONFERENCIA (IV)

Gracias al ILS (Internet Locator Server) podemos localizar a usuarios determinados para poder establecer una conferencia. Sin éste no podríamos determinar a qué persona queremos llamar, ni por tanto, comenzar una comunicación. Conoceremos cómo se utilizan los atributos extendidos y sus limitaciones.

WWW



XML (III). El lenguaje XSL

En el presente artículo analizaremos dos de las formas que tenemos para presentar los datos XML contenidos en una página HTML. Hablamos de las hojas de estilo en cascada (CSS, Cascade Style Sheet) y del lenguaje XSL (Extensible Stylesheet Language); además haremos una introducción sobre este último, un estándar realmente potente que complementa a la tecnología XML.

Windows 95/98



Programación del registro de Windows (I)

El objetivo principal de esta serie consiste en aprender la forma de acceder al registro de Windows desde nuestros propios programas. Todo ello lo haremos desde la API de Windows y con Visual Basic.

Nuevas tecnologías



DirectX 6.1 (III)

Direct3D es la parte de DirectX encargada del proceso gráfico en 3 dimensiones. Aprovecha las últimas tecnologías para representar con fidelidad entornos virtuales. Para evitar complicaciones vamos a explicar el manejo con la creación de un device por defecto, con la asignación de color, luces y rotación del objeto.

Herramientas

VISUALAGE C++ 4.0

Aunque esta herramienta no se encuentra entre las más conocidas para desarrollar en C++, su última versión ofrece todo un conjunto de revolucionarias características y nuevos conceptos que seguro que gustarán a más de uno; sobre todo por su conjunto de asistentes y herramientas enfocadas al desarrollo de aplicaciones multiplataforma orientadas a objeto.

68 Bases de datos

DESARROLLO CLIENTE/SERVIDOR (II)

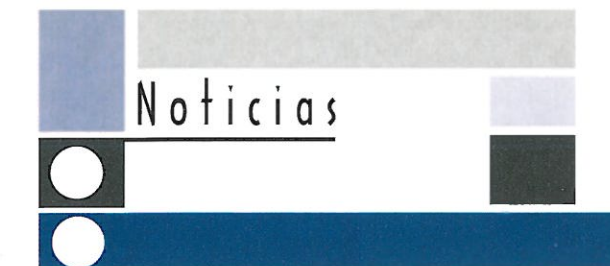
A partir de este mes comenzamos con un ejemplo de desarrollo de un sistema cliente/servidor en dos capas. Esta entrega se ocupa del desarrollo del cliente. La herramienta elegida ha sido Visual Basic 6.0 y la base de datos utilizada Access.

Libros

La informática está de moda. Tal vez por eso no dejan de aparecer manuales de interés. En esta ocasión os reseñamos los cuatro que nos han parecido más interesantes, destacando la Biblia de Delphi 4.

Ayuda al lector

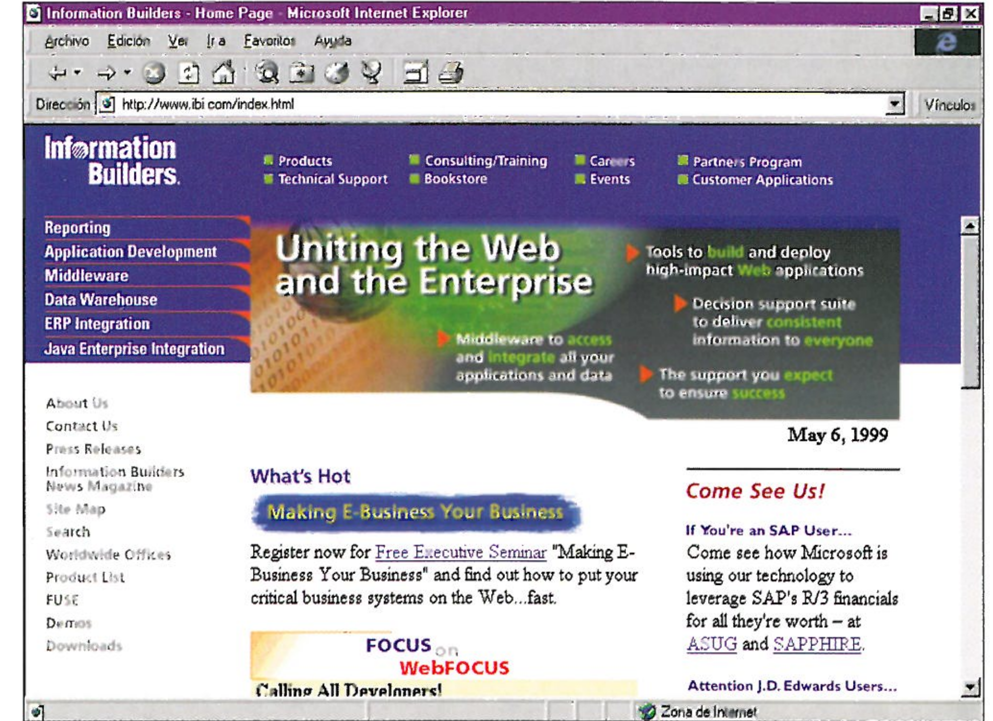
Como siempre incluimos aquellas preguntas que nos han parecido más interesantes para que podáis resolver todas vuestras dudas.



INFORMATION BUILDERS SE UNE A LA ALIANZA DATA WAREHOUSING

Information Builders ha pasado a formar parte de la alianza *Data Warehousing Alliance* (DWA) de *Microsoft*. El objetivo es ayudar a los clientes a construir, utilizar y gestionar aquellas soluciones de *data warehousing* que se integran completamente con las plataformas *Windows NT* y *SQL Server 7*.

SmartMart, la conocida familia de herramientas de esta empresa, es una completa solución de *data warehouse* que utiliza los drivers de datos más potentes y versátiles de la industria para extraer información desde los ficheros fuente, incluyendo a las BBDD relacionales y no relacionales.



Los datos extraídos son utilizados posteriormente por los componentes de transformación, creación de bases de datos y directorios de información de *SmartMart* para construir un nuevo *data warehouse*.

Además esta herramienta cumple los diferentes requisitos técnicos de *Microsoft*, como la integración con su repositorio, soporte OLE DB y la integración con SQL Server 7, entre otros. Podéis encontrar información adicional en **www.ibi.com**

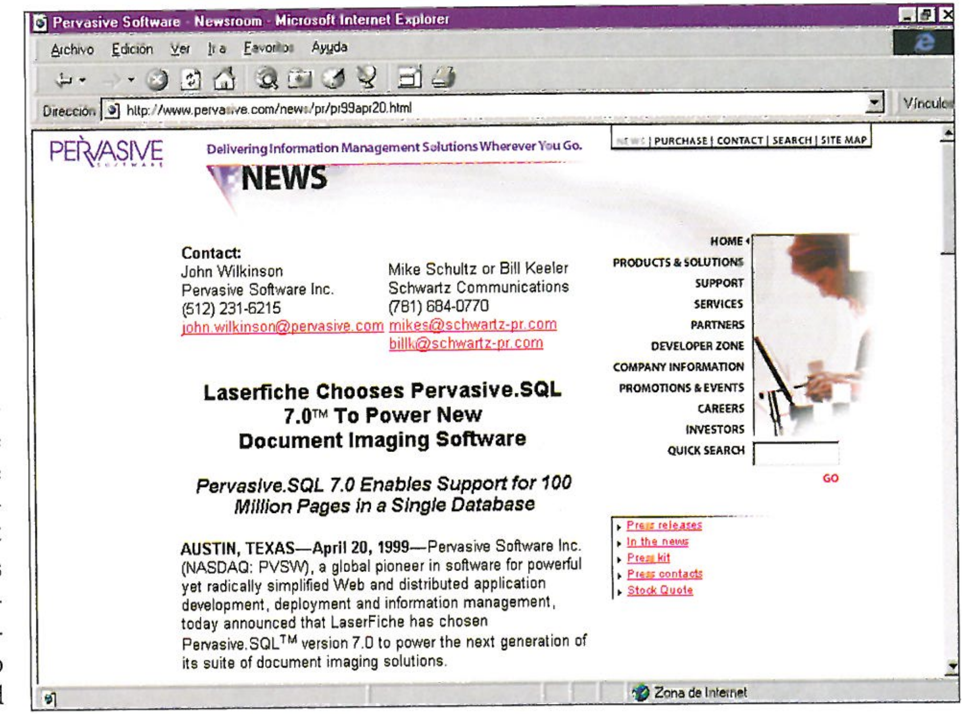
EL P7 FINANCIALS TIENE NUEVA VERSIÓN

La empresa *On Line* sigue buscando soluciones para la empresa industrial. En este caso es la nueva versión del *P7 Financials*, un módulo contable-financiero que ha sido adaptado a la normativa contable de todos los países. Este programa permite realizar la contabilidad analítica por centro de coste o subcuentas y permite la simulación de tantos cierres de ejercicio como se desee. Gestiona hasta 6 niveles de impuestos así como el inmovilizado, permitiendo la simulación de distintos métodos de amortización. Además dispone una aplicación para gestionar la tesorería que, como novedad, incluye la posibilidad de llevar la contabilidad en dos monedas indistintamente, como por ejemplo euros y pesetas. De tal forma que en un solo sistema se recoge todo el ciclo logístico industrial.

En un principio *P7 Financials* está diseñado para las PYMES con un volumen de negocios entre 500 y 8.000 millones de pesetas, aunque también puede utilizarse en grandes empresas.

PERVASIVE.SQL 7.0 IMPULSA UN NUEVO SOFTWARE DE TRATAMIENTO DE IMÁGENES

La compañía *Pervasive* anuncia que *LaserFiche Document Imaging* ha elegido *Pervasive.SQL* versión 7.0 para impulsar la próxima generación de su gama de soluciones de tratamiento de imágenes de documentos.



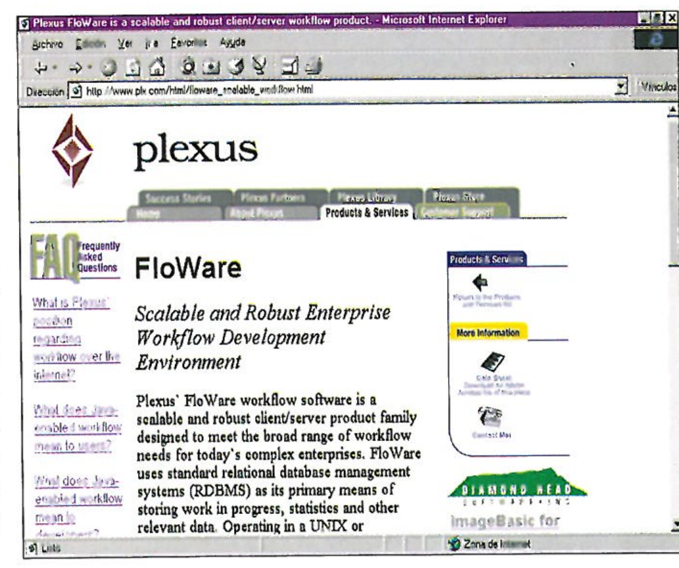
Esta última empresa está especializada en ofrecer soluciones de gestión integradas, inteligentes, flexibles y fáciles para una amplia gama de necesidades de empresa y de gobierno. Su *software* convierte en el *hardware* PC existente en bases de datos robustas de búsqueda y recuperación que pueden escalarse fácilmente desde un único usuario hasta una gran red corporativa.

La próxima generación de la gama de tratamiento de imágenes de documentos incluye *LaserFiche WebLink* 4.3 y el núcleo de *software* de la compañía *LaserFiche* 4.3, utilizado por numerosas organizaciones privadas y públicas. Puedes ampliar la información en **www.pervasive.com**

BANCTEC ANUNCIA LA DISPONIBILIDAD DE FLOWARE 5.0

Acaba de ser presentado *FloWare 5.0*, última versión de la línea de productos de *Workflow* más escalable del mercado. El principal objetivo para el que ha sido diseñado es el de satisfacer las numerosas necesidades de negocios críticos que tienen las complejas empresas de hoy.

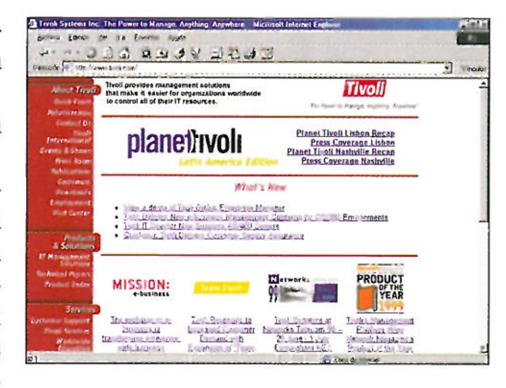
Este nuevo producto ofrece muchas características que facilitan la integración con aplicaciones Internet y *middleware*. Entre éstas cabe destacar, la integración con LDAP, *User Emulation Login* que adopta temporalmente el perfil de un usuario que está en periodo de inactividad, y *Guest-User Support* que permite al sistema seguir e identificar a usuarios esporádicos. *FloWare* utiliza sistemas de administración de base de datos relacionales estándares como su medio inicial de almacenaje de trabajo en curso. Existe más información acerca de éste y otros productos en **www.banctec.com**



SÓLO PROGRAMADORES

TIVOLI REVOLUCIONA LA GESTIÓN DE ALMACENAMIENTO

La compañía *Tivoli* ha anunciado su tercera iniciativa por Integridad de la Información que va destinada a resolver los retos planteados por la gestión de almacenamiento en las economías digitales actuales.



Esta iniciativa permitirá a las organizaciones aunar las prácticas comerciales claves para la empresa con las políticas de gestión de almacenamiento, permitiendo a los clientes, por primera vez, emplear la información para gestionar la empresa en lugar de emplearla como un simple soporte de las operaciones. Puedes ampliar la información en **www.tivoli.com**

UCON 99. UN IMPORTANTE ENCUENTRO PARA DESARROLLADORES WEB Y MULTIMEDIA

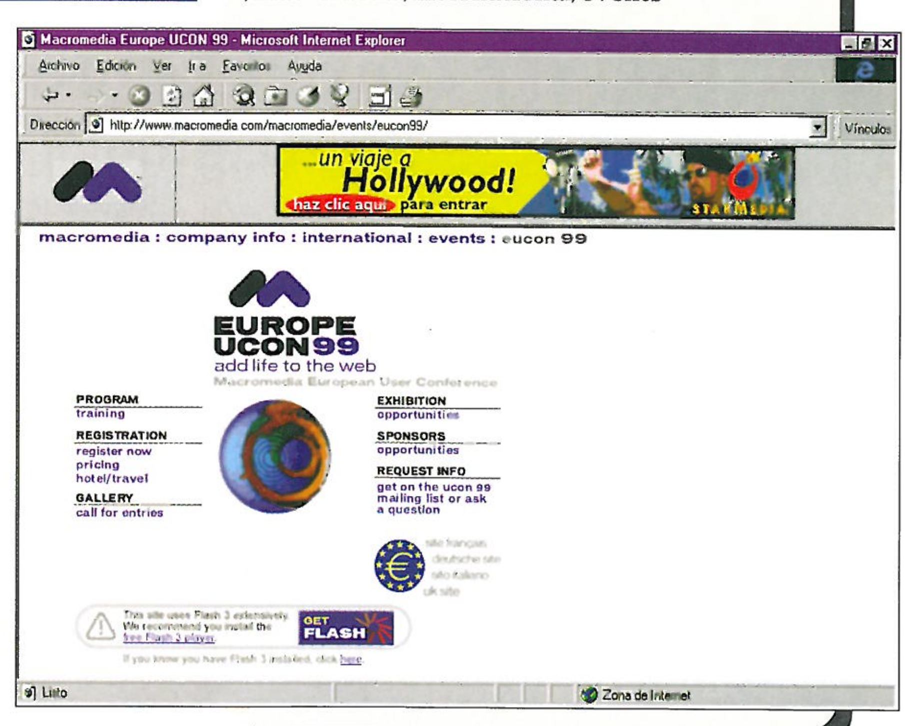
Los días 24 y 25 de junio ha tenido lugar en Disneyland París la celebración de Macromedia Europe UCON 99. Un evento anual que ofrece a los desarrolladores Web y multimedia, así como a los creativos profesionales, la oportunidad de encontrar nuevas tendencias y abrir perspectivas de trabajo.

Durante estos dos días se han impartido numerosas conferencias a las que asistieron los mejores técnicos especialistas de Macromedia, los más prestigiosos profesionales del sector y también los más conocidos gurús de la Web. Además se ofrecieron diversas demostraciones técnicas y algunas charlas sobre formación *on line*.

Los interesados en el tema pueden contactar por E-mail en **eucon@macromedia.com** o a través de Internet en la dirección **www.macromedia.com /macromedia/international/events**

SUNPEAK YA ESTÁ LISTO

Sun Microsystems ya tiene lista su nueva estructura de tecnología de la información, gracias a la colaboración de *Andersen Consulting*. Se trata de *SunPeak*, un proyecto interno de reingeniería que permite la migración de la estructura informática basada en sistemas *mainframe* a una plataforma exclusiva de Sun, flexible, escalable y optimizada para la economía en red. Todo el proyecto, implementado en fases, supone un sistema de compartición de la información basado en el modelo *"publish and suscribe"*, donde cada aplicación de *software* dispone de una interfaz exclusiva, la propia autopista de la información.



LA PROGRAMACIÓN EN CLIPPER PARA WINDOWS TIENE NUEVA HERRAMIENTA

La empresa española *Fivetech* acaba de lanzar la nueva versión de su entorno de programación en Windows para los usuarios de *CA-Clipper*. Se trata de *Fivewin* 2.0, con un acceso completo a cualquier API de 32 bits, manejo de tablas ODBC bajo el standard RDD de CA-Clipper y nuevas clases para manejo de *Sockets* (Internet). Todo ello con el estilo de *Microsoft Office*. *Fivewin*, que se utiliza como herramienta básica de programación, es el standard de programación en lenguaje *Clipper* bajo el entorno *Windows,* en todas sus versiones, porque permite migrar sus actuales aplicaciones Ca-Clipper & MS-DOS a este entorno. Podréis encontrar más información en **www.fivetech.com**

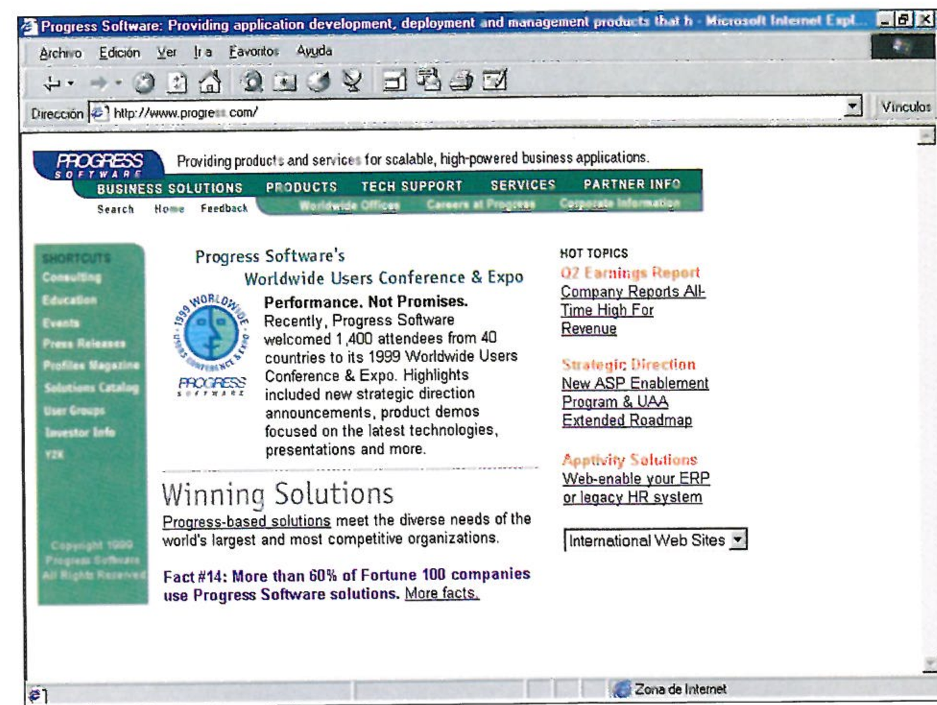
NEWELL RUBBERMAID IMPLANTA MOVEX

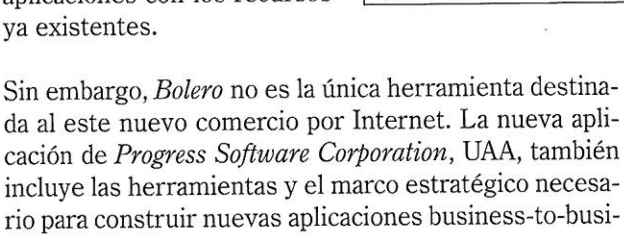
Tras la firma del acuerdo con *Intentia*, la recién fusionada compañía *Newell Rubbermaid* implantará en todas sus subsidiarias europeas la aplicación integrada *Movex.* Esta decisión se ha debido, según fuentes de la empresa, a las amplias funcionalidades que aporta en las áreas claves de distribución, logística y producción. El contrato entre ambas empresas abarca un total de 22 empresas en 14 países con un valor de 1.100 millones de pesetas.

LAS EMPRESAS APUESTAN POR EL COMERCIO ELECTRÓNICO

El comercio electrónico se está desarrollando a gran velocidad. Buena prueba de ello son las nuevas aplicaciones que diferentes empresas están desarrollando en los últimos tiempos.

Software AG ya ha presentado *"Bolero"*, la primera herramienta de desarrollo de aplicaciones específicamente diseñada para el comercio electrónico. Pero también para gestionar las relaciones y operaciones comerciales entre las empresas utilizando Internet. Todo ello para que las empresas se adapten al nuevo entorno empresarial pero integrando las nuevas aplicaciones con los recursos ya existentes.

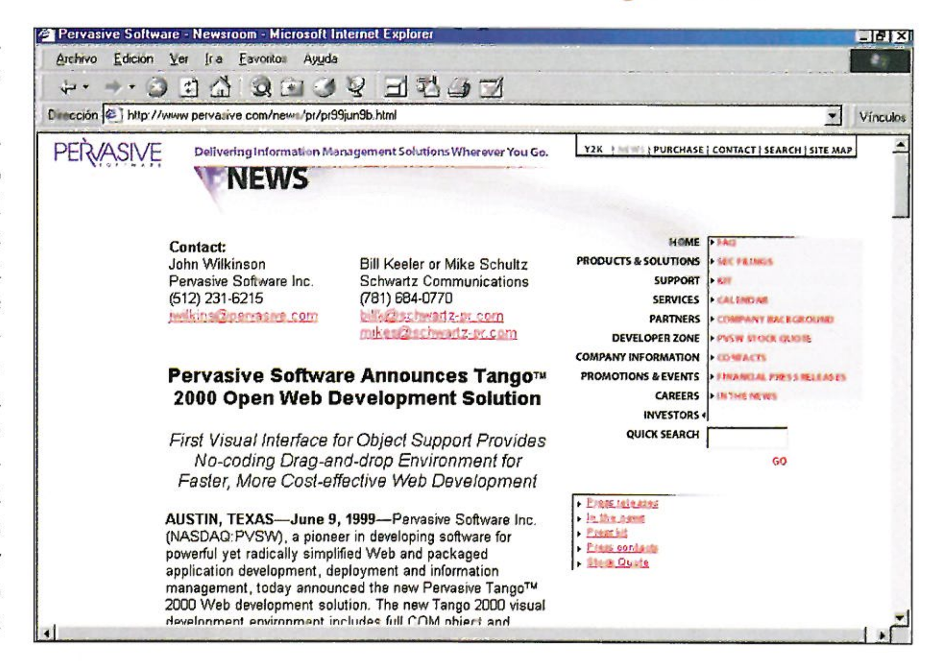



Sin embargo, *Bolero* no es la única herramienta destinada al este nuevo comercio por Internet. La nueva aplicación de *Progress Software Corporation*, UAA, también incluye las herramientas y el marco estratégico necesario para construir nuevas aplicaciones business-to-business que operan en la Red. Esta nueva aplicación cuenta con el servidor de aplicaciones *Apptivity,* denominado con el código *"Vader"*, que combina un modelo de programación basado en componentes, intercambio de datos basado en estándares y un sistema de mensajería de negociado preparado para Internet.

EL DESARROLLO DE PÁGINAS DE DREAMWEAVER SE REVOLUCIONA CON TANGO OBJECTS

Pervasive anuncia la disponibilidad de su producto *Tango Objects* para *Dreamweaver2*.

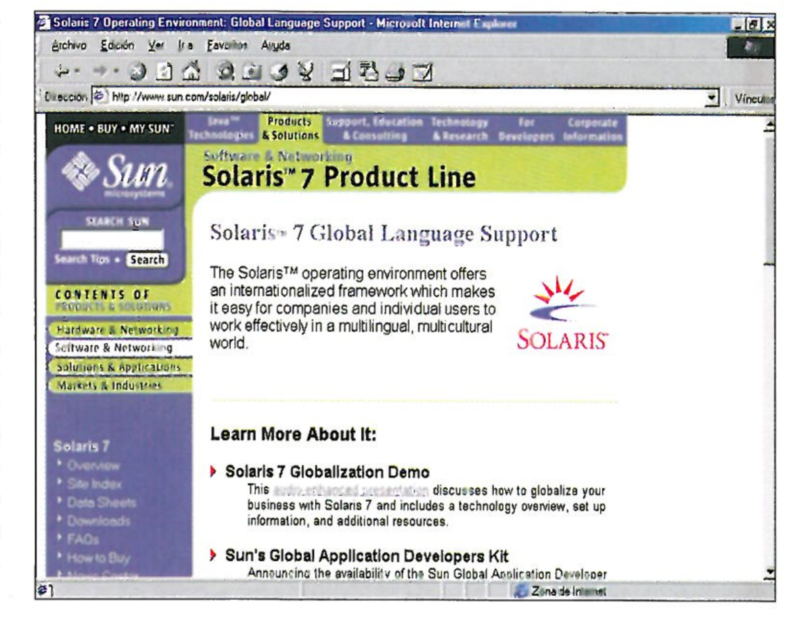
Los desarrolladores de aplicaciones Web que utilizan Tango se benefician de la adición del entorno de desarrollo de páginas *Dreamweaver* que permite la creación y gestión visuales de éstas. Por otra parte los desarrolladores que trabajan con *Dreamweaver2* se beneficiarán de la potente interfaz de *Tango Pervasive* para añadir capacidades sofisticadas de codificación HTML y de gestión de datos a sus aplicaciones sin necesitar un avanzado conocimiento de *SQL* u otras técnicas de codificación.



La integración ofrece a los usuarios una mayor funcionalidad y acelera los ciclos de desarrollo para poder comercializar sus productos más rápido y conseguir diseños más económicos. Más información en **www.pervasive.com**

NUEVO SISTEMA DE INTERNACIONALIZACIÓN PARA SOLARIS

El nuevo kit de *Sun Microsystems* para el Desarrollo de Aplicaciones Globales para el entorno operativo *Solaris* ya está disponible. Está dirigido a desarrolladores de *software* corporativo y a fabricantes independientes de *software* que necesiten aplicaciones multilingües, sobre todo porque *Solaris* se distribuye en 37 idiomas. El proceso de este kit consiste en diseñar aplicaciones con un idioma neutro que permite a los clientes introducir, visualizar y generar textos en diversos idiomas. Siempre es mucho más fácil, y barato, adaptar el producto a determinadas zonas o mercados geográficos añadiendo contenidos específicos, que diseñar una aplicación para esa zona o mercado. Además el kit incluye completas herramientas de internacionalización y la documentación necesaria para solucionar gran parte de los problemas de diseño y desarrollo. Más información en **www.sun.es**





HERRAMIENTAS DE PROGRAMACIÓN

ENTORNOS DE DESARROLLO

ACTIVEPERL 517

Kit de desarrollo de Perl. Incluye los módulos Perl para Windows 32 bits, Perl para ISAPI, PerlScript y una herramienta adicional Perl Package Manager.

INTERNET PROGRAMMING LANGUAGE (IPL)

Lenguaje similar al C que permite el desarrollo de aplicaciones para Internet. Incorpora funciones para el envío de E-mail SMTP, transferencia de ficheros, creación de formularios CGI,.sockets para conexiones TCP/IP, etc.

PALM-SIZE PC SDK 1.2

Kit de desarrollo para PDA's con Windows CE. Ofrece soporte para desarrollar aplicaciones usando Visual Basic 6.0 y Visual C++ 6.0. La interfaz de emulación permite ejecutar las aplicaciones Windows CE en máquinas Windows NT.

SCRIBBLER 98 V1.8.0

Editor para programadores principiantes y avanzados que facilita la creación y verificación de código DHTML, JavaScript o VBScript. Incluye una librería con 30 scripts predefinidos.

SYNEDIT 0.50 BETA

Editor de texto que incluye un editor binario y hexadecimal. Puede trabajar con compiladores y otras herramientas de línea de comandos, guardar claves y frases utilizadas frecuentemente y salvar los ficheros en formato RTF.

VISUAL PROGRAMMING ARMOURY 1.23

IDE multipropósito para C++ y Java, que permite gestionar grandes proyectos de programación de manera muy sencilla. Ofrece diseño visual de formularios, mapas y bases de datos.

XML SPY 2.5

Potente editor de XML y DTD. Ofrece una potente visualización de la estructura, distintas opciones de impresión, número ilimitado de acciones *deshacer* y capacidades para buscar y reemplazar código.

LENGUAJES

VISUAL BASIC

CODESMART 3.3

Nuevas funcionalidades para Visual Basic 5 o superior. Ofrece ítems de código coloreados, , una base de datos actualizable de código, un sistema de creación de macros, etc.

PCTALKIE 2.5

ActiveX que añade reconocimiento de voz a aplicaciones VB. Puede controlar la velocidad de la locución, la calidad de la pronunciación y convertir el texto generado en ficheros WAV.

VOICEACTION 1.2.500

Control ActiveX que proporciona reconocimiento de voz. Ofrece una interfaz muy sencilla que proporciona al usuario la posibilidad de construir su propia base de datos de vocabulario y diseñar, mediante inteligencia artificial, una base de datos de palabras.

JAVA

GATE KEEPER 3.2

Aplicación JavaScript que restringe el acceso a determinadas páginas Web. Para ello emplea claves de acceso que el usuario deberá introducir para poder consultar las páginas protegidas. Trabaja sin utilizar comandos CGI.

GUARDIAN 1.1

Sistema de registro de software en desarrollo Java. Permite crear versiones shareware de programas con límites de ejecución en un determinado periodo.

RIADALOCK 1.02

Herramienta Java que permite restringir el acceso, a determinadas áreas de nuestro sitio Web, a usuarios autorizados. Trabaja con la mayoría de los servidores de Webs y no necesita scripts CGI-BIN.

HTML

CSS WIZ 0.11

Generador de hojas de estilo en cascada. Utiliza asistentes para modificar los atributos y generar el código de la hoja de

estilo, que es copiado en el fichero HTML.

ECBUILDER 4.00.205

Utilidad para crear sitios Web comerciales para promoción y venta de productos y servicios. Soporta 35 tipos de ficheros gráficos y puede importar ficheros de texto, sonido y vídeo. Ofrece seguridad SSL.

METABOT PRO 1.0A

Automatiza la creación, inserción y verificación de metatags HTML. Soporta más de 50 distintos tipos de metatags.

WORDTOWEB 2.0

Convierte documentos complejos de Microsoft Word en publicaciones HTML.

OTROS

FREEAMP SOURCE CODE 1.2.3

Código fuente del reproductor de audio FreeAmp. Incluye los archivos de proyecto generados con MS Visual C++ 5.0. Ofrece soporte para ficheros M3U, listas de reproducción y servidores Shoutcast.

IE URL LINK CONTROL

Control ActiveX que emula el comportamiento de los enlaces de las URL´s de los navegadores de Internet para su inclusión en nuestras aplicaciones.

LIBCON 0.21

Librería C++ para el desarrollo de juegos. Manera sencilla de gestión, mediante la API, de las siguientes características: texturas 2D y 3D, tarjeta de sonido, teclado, ratón y joystick.

MULTIMEDIA CONVERSION LIBRARY 1.20

Añade a las aplicaciones capacidades de conversión de ficheros de imágenes y vídeo. Puede importar casi cualquier tipo de fichero gráfico. También soporta los formatos de vídeo AVI, FLC, FLI, HAV, M2V y MPG.

RASCONTROL 1.0

Control ActiveX que añade a una aplicación, el control remoto de la conexión a Internet, la desconexión, los accesos telefónicos, etc.

HERRAMIENTAS Y UTILIDADES

BISON/FLEX WIZARD 1.5

Herramientas GNU adicionales para MS Developer Studio 5.0. Permiten la creación de esqueletos y módulos de proyectos de desarrollo en la versión 5.0 de Microsoft Developer Studio. Se incluye el código fuente completo.

EVOLVABLE WEBTEST

Ejecución de tests de detección de fallos en el servidor Web. Para usar WebTest basta con visitar las páginas que se quieren comprobar y pulsar sobre el botón "Save".

MICROSOFT WINDOWS MEDIA ON-DEMAND PRODUCER 2.0 BETA

Permite la descodificación de ficheros WAV y AVI para su conversión a ASF. La aplicación incorpora un sistema visual que permite una manipulación rápida de los elementos.

MICROSOFT WINDOWS MEDIA TOOLS 4.0 BETA

Kit para la creación de audio comprimido. Este kit está especialmente indicado para su utilización en Internet o productos multimedia que necesiten de compresión de audio.

MILES SOUND SYSTEM 5.0

Potente kit de desarrollo de librerías de sonido. Soporta audio digital, Dolby, 3D audio, descompresión digital (MPEG Layer-3 y ADPCM), MIDI, CD-Audio (libro rojo), y otros.

REDREGISTRATION DEVELOPERS SDK 2.0A

Kit de desarrollo para la creación de aplicaciones limitadas en tiempo y en número de ejecuciones y usuarios. Trabaja con todos los entornos de desarrollo que soporten DLLs y controles ActiveX.

RESOURCE STANDARD METRICS (RSM) 5.0

Analiza el código C, C++ y Java para optimizarlo y conseguir certificaciones ISO9001, TickIt y SEI. Puede determinar los 50 mayores problemas de calidad (estilo y análisis) que no detectan los compiladores tradicionales.

TABLE2HTML 1.2

Convierte las listas de Word en una tabla HTML. Permite especificar el formato de la fuente, la justificación del texto, etc. Genera automáticamente el HTML.

REDES

LOCALES

NORTON GHOST 5.1C

Facilita la actualización e instalación de programas en una LAN. Crea una imagen virtual de la información del disco duro y que puede ser copiada a otros equipos creando instalaciones completamente idénticas.

WINPROXY LITE 2.1H

Facilita la conexión de los usuarios de una red LAN a Internet utilizando una

única cuenta, un módem y una dirección IP. Soporta los protocolos usados habitualmente en Internet.

DISTRIBUIDAS

APACHE HTTP SERVER 1.3.6

Versión para Linux de este potente y configurable servidor HTTP.

IBM VISUAL BUILDER PARA VISUALAGE WEBRUNNER SERVER WORKS

Permite a los usuarios de VisualAge for C++ la creación de aplicaciones servidor Web en Common Gateway Interface (CGI), Netscape Server (NSAPI) e Internet Connection Server (ICAPI).

INFRADIG PERSONAL SERVER 1.99

Programa servidor E-mail, Web y FTP. Soporta ESMTP, POP3, IMAP4, FTP, HTTP, CGI y ISAPI. Permite utilizar hosts virtuales, directorios y usuarios virtuales así como su autentificación.

ZPTSERVER 1.52

Servidor Web de gran potencia escrito completamente en Java. Ofrece gran número de opciones de configuración.

OTROS

EXPRESSFS SERVER 1.4

Servidor FTP muy fácil de instalar y configurar, que realiza conexiones anónimas o protegidas por password. Puede visualizar la dirección IP dinámica y mostrar en una ventana la actividad de los usuarios conectados.

NET LIGHTNING LITE 1.5

Almacena la información de las DNS del servidor en un fichero local acelerando la conexión a Internet.

SURFSTATS LOG FILE RETRIEVER AND ANALYZER 3.2

Genera 26 tipos distintos de informes basados en HTML que ofrecen detalles sobre las visitas recibidas en nuestro sitio Web.

IMPRESCINDIBLES

ANTIVIRUS

AntiViral Toolkit Pro 3.0.129
McAfee VirusScan 4.0.3
Panda Antivirus 6.06 Platinum

GRÁFICOS

Icon Bank 4.0 Gold Edition
IconForge 4.1
MainActor 3.02
Paint Shop Pro 5.01 en castellano
SureThing CD Labeler 2.0
ThumbsPlus 4.0

INTERNET

Añadir Pro 4.0
Copernic 99 v3.01
Cuentapasos 3.72
Dial-Up Magic 1.8
Eudora Light 3.0.6
GetRight 3.34
Guardián 1.1
HomeSite 4.01
ICQ 99a beta 2.21 build 1800
MIRC 32 5.60
Net Vampire 3.3
PGP 6.0.2i
WebTrends Enterprise Suite 3.5

MULTIMEDIA

AudioCatalyst 2.01
Cool Edit Pro 1.2
COWON Jet-Audio 4.02
CDH Media Wizard 3.7
Sonique 1.00
WinAmp 2.23

NAVEGADORES

Internet Explorer 5.0.2314.1003
Netscape Communicator 4.61
Opera 3.60

UTILIDADES

3DMark 99 Max b200
Babylon Translator 20.12
CDRWin 3.7c
DirectX 6.0 Castellano, DirectX 6.1 Inglés
Emergency Recovery System 8.66
Nero 4.0.1.3
Where Is It? 2.00
Windows Commander 3.53
WinZip 7.0

DOCUMENTACIÓN /TUTORIALES

LIBRARY-LYNX 1.02A

Colección de páginas Web que contienen enlaces para acceder a numerosos sitios Web con recursos para desarrolladores.

NEWS-LYNX 1.02A

Colección de páginas Web que contienen enlaces para acceder a numerosos sitios Web de información incluyendo periódicos y revistas.

SOFTWARE-LYNX 1.02A

Colección de páginas Web que contienen enlaces a docenas de sitios Web con programas gratuitos, shareware y versiones comerciales.

VBA TUTOR 2

Tutorial Visual Basic para Aplicaciones de Office 97. VBA Tutor ilustra, a través de documentos Microsoft Word, cómo se debe usar VBA junto con Office 97. Incluye ejercicios.

Multimedia con Java: sonido y vídeo (y III)

Javier Sanz Alamillo (jsanza@teleline.es)

En este último artículo se completará la exposición de los conceptos fundamentales de *Java Media Player (JMP)*, lo que permitirá al lector crear reproductores de audio y vídeo de una forma rápida y sencilla, sin la necesidad de amplios conocimientos en programación multimedia.

Continuando con los fundamentos de *Java Media Player* se describirán los estados de un reproductor, las características sobre la invocación de los métodos y cómo utilizar determinados componentes.

Un objeto Player tiene definidos seis estados, pero únicamente puede estar en uno de ellos en un determinado momento

Esto permitirá desarrollar un reproductor básico, gracias al cual, el lector comprobará cómo con escasas líneas de código y mediante un desarrollo sin complicaciones se puede construir un reproductor como los típicamente presentados en los entornos *Windows*.

ESTADOS DE UN OBJETO TIPO PLAYER

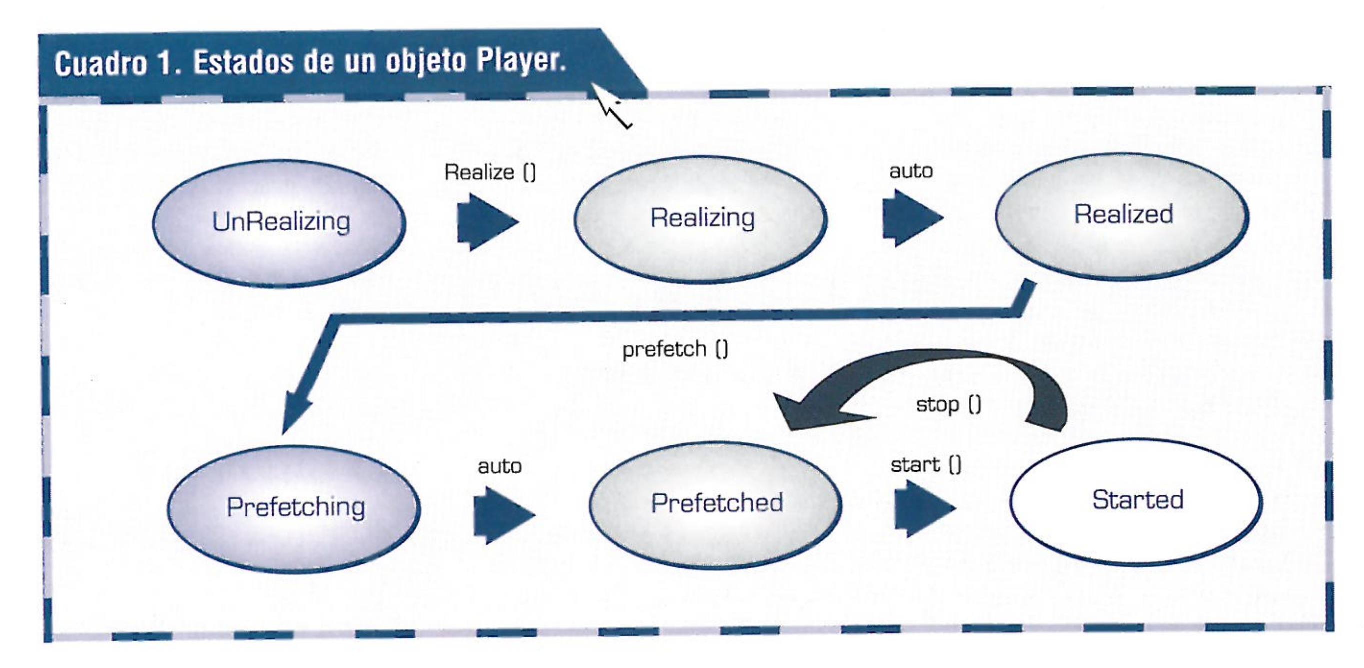
Un objeto *Player* tiene definidos seis estados, pero únicamente puede estar en uno de ellos en un determinado momento. Recordando la jerarquía de clases en la que se relacionan *Player, Controller y Clock*, tenemos que la definición de los estados pasa por el uso de la interfaz *Clock* que los agrupa y define dos estados primarios: *Stopped* y *Started* (parado e iniciado) Para permitir una mejor gestión de los recursos, la interfaz *Controller* divide a su vez el estado *Stopped* en los siguientes:

- *Unrealized*
- *Realizing*
- *Realized*
- *Prefetching*
- *Prefetched*

Con todo ello se obtienen los seis estados. A continuación abordaremos los detalles que caracterizan a cada uno de estos estados, ya que durante la reproducción multimedia un objeto *Player* pasa a través de ellos hasta llegar al estado *Started*:

- Un objeto *Player* en el estado *Unrealized* significa que está instanciado, pero que aún no tiene ningún control sobre los datos multimedia que gestionar. Cuando el objeto *Player* se crea, automáticamente pasa al estado *Unrealized*.
- Se pasa al estado *Realizing* desde *Unrealized*. Este estado se alcanza cuando el objeto *Player* determina qué tipo de recursos necesita, y durante este proceso, los adquiere los. Esto incluye el procesamiento

Cuadro 1. Estados de un objeto Player.



de los datos y de otros recursos de uso exclusivo. Un recurso de uso exclusivo es aquél que únicamente puede ser utilizado por un objeto *Player* a la vez, como determinados dispositivos *hardware* (tarjetas de sonido en algunas condiciones).

El método Deallocate() se encarga de liberar el dispositivo hardware que está en uso

- Cuando un objeto *Player* finaliza su estado *Realizing* pasa el estado *Realized*. En este estado, el objeto tiene disponibles todos los recursos para la reproducción y puede determinar qué tipo de datos está manejando. Gracias a esto, se pueden determinar una serie de controles y componentes visuales. Por ejemplo, si se determina que los datos son de un formato de vídeo se puede disponer de una barra de control con botones del estilo *play, stop, pause*, etc.
- Del estado *Realized* se pasa a *Prefetching*, en el cual el objeto *Player* está preparado para reproducir el tipo de datos multimedia manipulado. En este proceso, el objeto *Player* realiza una mínima precarga de los datos, obteniendo así la posesión de un recurso de uso exclusivo y de todo lo necesario para iniciar la reproducción de los datos. Este estado se repetirá en el caso de que se altere el orden de reproducción o el reproductor requiera del *buffer* información adicional para procesar los datos.
- Cuando un objeto *Player* finaliza su estado *Prefetching* pasa al estado *Prefetched*. Un objeto *Player* en este estado está preparado para comenzar la reproducción. A partir de este estado puede pasar a situarse en el estado *Started*.
- Cuando se ejecuta una sentencia *Start()* un objeto del tipo *Player* se sitúa formalmente en el estado *Started*. En este estado, el objeto *Player* dispone de todos los medios necesarios para iniciar la reproducción y únicamente está esperando una determinada condición para comenzar la reproducción.

Los objetos *Player* crean eventos del tipo *TranstionEvents* al pasar de un estado a otro. La interfaz *Controller Listener* permite controlar en los programas en qué estado está el objeto *Player* y así responder a determinados eventos adecuadamente. Mediante este sistema de eventos, los objetos *Player* pueden controlar por ejemplo, la latencia de reproducción, mediante el control de los estados *Realizing* y *Prefetching*. También permite asegurar que el estado de un objeto *Player* es un estado válido antes de ejecutar determinados métodos del mismo.

MÉTODOS DISPONIBLES EN CADA ESTADO DE PLAYER

Para evitar determinadas condiciones de carrera que puedan surgir en la reproducción multimedia, algunos métodos de los objetos *Player* no pueden ser invocados en determinados estados. En la tabla 1 se muestran las restricciones determinadas en *JMP*. Si el programador ejecuta una llamada ilegal en un determinado estado de *Player*, se lanzará una excepción o un error.

LLAMADA A LOS MÉTODOS DE JMP

Las excepciones y errores que se producen al usar *JMP* se pueden clasificar de la siguiente manera:

- Los *Java Media Errors* son lanzados cuando un programa ejecuta un método que es ilegal en el estado actual de un objeto *Player*. Los objetos tipo *Error* son lanzando en situaciones donde se debe realizar el control sobre un determinado estado y la operación solicitada puede ser ejecutada en determinadas condiciones de carrera. Por ejemplo, no es correcto llamar a ciertos métodos cuando el objeto *Player* está en el estado *Started*. Es tarea del programador asegurarse de que el objeto *Player* es parado antes de usar esos métodos. Las aplicaciones no deberían capturar las excepciones mediante *try/catch* de los objetos *Error*. Una aplicación correctamente desarrollada nunca deberá tener en cuenta estos tipos de circunstancias.
- Las *Java Media Exceptions* son lanzadas cuando un programa llama a métodos que no pueden ser totalmente ejecutados o no son aplicables según el estado actual del objeto *Player*. Las excepciones son lanzadas en situaciones donde el programador no puede tener control sobre el estado del *Player*. Por ejemplo, una si se intenta sincronizar dos objetos *Player* con ciertas incompatibilidades. Esto no se define como un error porque el programador no puede determinar ciertas circunstancias. Igualmente, si se ejecuta un método que sólo es aplicable en el estado *Started* y el objeto *Player* se encuentra en el estado *Stopped*, se lanza una excepción.

En el estado Started el objeto Player está preparado para comenzar la reproducción

Algunos métodos de *JMP* devuelven valores que indican el resultado de la ejecución del método. Por ejemplo, comprobando los valores de retorno se puede determinar lo que está ocurriendo en la reproducción. Los valores de retorno indican:

- El valor actualmente determinado. Por ejemplo, no todos los objetos *Player* pueden representar un tipo de datos en un determinado *ratio*. Si se utiliza el método *SetRate(5.0)* el objeto *Player* intentará situar su *ratio* de reproducción lo más próximo a **5**, por lo que al comprobar el *ratio* debería devolver el número determinado.
- La información que se solicita no está disponible. Por ejemplo, no se puede llamar al método *GetDuration* hasta que el objeto *Player* no haya comenzado la reproducción, ya que devolvería un valor *DURATION_UNKNOWN*. Si es llamado durante la reproducción, devuelve la duración actual del mismo.

CREANDO UN REPRODUCTOR DE VÍDEO

Una vez que ya se han mostrado los fundamentos sobre *Java Media Player*, se va a mostrar como construir un reproductor de vídeo en formato *MPEG*. Mediante este reproductor se podrá reproducir cualquier otro formato de datos, tanto de audio como vídeo.

La creación del reproductor se basará en un *applet*, en el cual se visualizarán unos controles y una ventana de visualización. Cuando se crea una aplicación, el programador es responsable de visualizar los componentes.

Tabla 1. Restricciones determinadas en JMP.

Método	Unrealized	Realized Player	Prefetched Player	Started Player
GetStartLatency	NotRealizedError	Legal	Legal	Legal
GetTimeBase	NotRealizedError	Legal	Legal	Legal
SetMediaTime	NotRealizedError	Legal	Legal	Legal
SetRate	NotRealizedError	Legal	Legal	Legal
GetVisualComponent	NotRealizedError	Legal	Legal	Legal
GetControlPanelComponent	NotRealizedError	Legal	Legal	Legal
GetGainControl	NotRealizedError	Legal	Legal	Legal
SetStopTime	NotRealizedError	Legal	Legal	StopTimeSetError
SyncStart	NotPrefetchedError	NotPrefetchedError	Legal	ClockStartedError
SetTimeBase	NotRealizedError	Legal	Legal	ClockStartedError
Deallocate	Legal	Legal	Legal	ClockStartedError
AddController	NotRealizedError	Legal	Legal	ClockStartedError
RemoveController	NotRealizedError	Legal	Legal	ClockStartedError
MapToTimeBase	ClockStoppedException	ClockStoppedException	ClockStoppedException	Legal

CREACIÓN DEL REPRODUCTOR

La creación del reproductor en un *applet* es una tarea en la que se utilizan sencillos métodos que hacen que construir este tipo de aplicaciones multimedia sea una tarea bastante asequible. El proceso se basa en las siguientes fases:

- La llamada al método *Init()* crear un objeto *Player* que utiliza un objeto *URL* en el cual se determina el tipo de datos que reproducir, en este caso un vídeo *MPEG*. Además se añade la gestión de eventos mediante el método *ControllerUpdate* en el cual se atenderán todos los eventos relacionados con la reproducción.
- Se ejecuta el método *Start()*, mediante el cual el reproductor es inicializado.
- Cuando se ejecuta *Stop()*, finaliza la reproducción y se eliminan los recursos adquiridos.
- Y al llamar al método *Destroy*(), se elimina el objeto *Player* y todos sus recursos.

En el siguiente código se puede ver un reproductor de vídeo básico. A continuación se va a detallar su funcionamiento para poder incluir más adelante ciertas posibilidades.

Pueden ocurrir determinadas condiciones de carrera si los métodos invocados no son válidos en función del estado

En la figura 1 se puede apreciar la salida de la reproducción multimedia de un vídeo *MPEG*.Como se puede comprobar, un reproductor elemental apenas cuenta con una decena de líneas de código y éste tiene las mínimas complicaciones. Veamos su funcionamiento con más detalle.

Listado 1. Código correspondiente al applet del reproductor.

```
import java.applet.*;import java.awt.*;import java.net.*;import
javax.media.*;public class reproductor
   extends Applet implements ControllerListener {
Player repro = null ;
  public void init () {
              setLayout(new BorderLayout());
     try {
      URL miurl = new URL ( getCodeBase(), "video.mpg");
      repro = Manager.createPlayer( miurl );
      repro.addControllerListener(this);
     }
  catch (Exception e ) {
          System.err.println ( e );
     }
  }
  public void start() {
              repro.start();
  }
  public void stop() {
              repro.stop();
     repro.deallocate();
  }
     public void destroy() {
              repro.close();
  }
  public synchronized void controllerUpdate(ControllerEvent evt
) {
              if ( evt instanceof RealizeCompleteEvent ) {
                 Component comp;
                 if ( (comp= repro.getVisualComponent()) !=
null ) {
                      add ("Center",comp);
        }
     }
  }
}
```

INICIALIZACIÓN DEL REPRODUCTOR

Cuando un *applet* comienza a ejecutarse, el método *Init* es llamado automáticamente. Al sobreescribir este método, el *applet* puede ser inicializado. En este procedimiento se realizan las tareas de carga y creación de los objetos y recursos necesarios para la reproducción.

Mediante la creación del objeto **URL miurl** se cargan los datos multimedia que reproducir, en este caso un vídeo *MPEG*. La razón de usar un objeto *URL* es que permite tanto la carga local de ficheros como la carga a través de la red *(Internet)*.

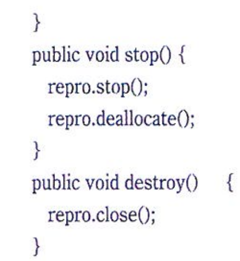
Se crea un objeto *Player* que será el encargado de realizar la reproducción. Para finalizar se registra el *applet* para poder controlar los eventos que se produzcan relacionados con el reproductor. Mediante este control se puede verificar el cambio de estado del mismo.

```
try {
  URL miurl = new URL ( getCodeBase(),
        "video.mpg");
  repro = Manager.createPlayer( miurl );
  repro.addControllerListener(this);
 }
catch (Exception e ) {
  System.err.println ( e );
}
```

CONTROLANDO LA REPRODUCCIÓN

A continuación se han definido los métodos *Start()* y *Stop()* que son llamados automáticamente cuando el *applet* comienza o es detenido. Se invocan los respectivos métodos del objeto *Player*, *Start()* y *Stop()*, teniendo en cuenta que es necesario invocar al método *deallocate()* para liberar el dispositivo *hardware* y algunos de los recursos adquiridos por el objeto reproductor, para permitir, por ejemplo, que otro objeto *Player* pueda iniciar su reproducción.

```
public void start() {
  repro.start();
```



```
}
public void stop() {
  repro.stop();
  repro.deallocate();
}
public void destroy()    {
  repro.close();
}
```

Cuando el *applet* finaliza, se eliminan los recursos que tenía asignados. El método sobreescrito *Destroy()* es invocado y es ahí donde se invoca al método *Close()* del objeto *Player* que libera todo los recursos obtenidos y lo elimina definitivamente.

GESTIÓN DE LOS EVENTOS

El *applet* registra los eventos como un *ControllerListener* en el método *Init()* del *applet*, pudiendo así recibir los eventos producidos. Para gestionar esos eventos se debe implementar el método *controllerUpdate* que es llamado automáticamente cuando se produce un evento. En este ejemplo, se comprueba el evento *RealizeCompleteEvent*, ya que en ese momento se podrán añadir por ejemplo controles de visualización y el componente que se encarga de la reproducción. Los componentes de la interfaz de usuario no pueden ser visualizados hasta que el objeto *Player* se encuentre en el estado *Realized*. Si el estado fuera *Unrealized* no se dispondría de la suficiente información sobre los datos que reproducir para asignar el componente adecuado. Por ello, el objeto *Player* debe esperar hasta que se produzca el evento *Realize CompleteEvent* y entonces visualizar los componentes requeridos.



Figura 1. Reproductor multimedia de vídeo.

VISUALIZACIÓN DEL REPRODUCTOR Y DE CONTROLES

JMP especifica un modelo de control de tiempo (*timing*) y reproducción cuando se visualizan datos multimedia, pero la interfaz de los componentes para los objetos *Player* utiliza las funciones definidas en el paquete *java.awt*. Por tanto, un reproductor tiene dos tipos de componentes del *AWT* para realizar la visualización, un componente visual y un componente de control.

PRESENTACIÓN DEL COMPONENTE DE VISUALIZACIÓN

Todo programa que requiere visualización necesita de un componente visual en el cual se realiza la reproducción. Incluso un reproductor de audio debería tener un componente visual, como un reproductor de ondas o una imagen animada. Para presentar un componente visual se deben seguir los pasos siguientes:

1. Obtener un componente mediante la llamada desde el reproductor al método *getVisualComponent()*.
2. Añadir al espacio de visualización del *applet* o aplicación el componente creado.

Las aplicaciones nunca deberán capturar excepciones Java Media Error

Las siguientes líneas de código permiten insertar el componente visual en un *applet* o aplicación.

```
Component comp;
if ( (comp=
```

```
    repro.getVisualComponent())
    != null ) {
  add ("Center",comp);
}
```

VISUALIZACIÓN DE CONTROLES DE REPRODUCCIÓN

La visualización de controles de reproducción permite al usuario tener un control sobre la misma. Por ejemplo, un reproductor debería tener un conjunto de botones del estilo *start, stop, pause*, y una barra para controlar el volumen. Cada objeto *Player* permite utilizar un componente de control por defecto. Para ello, se debería llamar al método *GetControlPanelComponent()* y añadir el componente al *applet* o aplicación para ser visualizado. En las siguientes líneas se muestra cómo añadir el componente de control por defecto en un reproductor:

```
Component comp ;
if ((comp=
    repro.getControlPanelComponen
    t()) != null ) {
  add ("South",comp);
}
```

Se pueden personalizar las opciones de control que se deseen presentar al usuario accediendo a las interfaces que definen el panel estándar. El componente de control permite la interacción con el reproductor y los controles propiamente definidos. Por ejemplo, los botones de *start* y *stop* del reproductor pueden ser utilizados a gusto del usuario y así por ejemplo, en la reproducción de vídeo se podría modificar el brillo y el contraste de la reproducción. Para poder realizar estas tareas se debe utilizar un objeto del tipo *Control*. Un objeto *Player* puede tener muchos tipos de objetos del tipo *Control* que definen los controles, las características de la interfaz de usuario. Para determinar los diferentes controles disponibles se utiliza el método *GetControls* con un objeto tipo *Player*. Si se desea averiguar qué tipo de soporte está determinado para la interfaz *CachingControl*, utilizado para mostrar el proceso de carga de los datos multimedia, se puede aplicar el siguiente código:

```
Player miplayer =
    Manager.createPlayer(url);
CachingControl ccontrol ;
Control[] controles =
    miplayer.getControls();
for (int cont = 0; cont < con-
    trols.length ; cont ++) {
  if ( controles[cont] instance-
    of CachingControl) {
      ccontrol = (CachingControl)
    controles[cont];
  }
}
```

Obsérvese que en función del la implementación utilizada se podrán utilizar determinados componentes y otros no, ya que algunos pueden no estar implementados.

VISUALIZACIÓN DEL COMPONENTE GAIN CONTROL

El componente *GainControl* se deriva de la interfaz *Control* y permite un manejo de las características del audio. Para utilizar este control se debe invocar al método *GetGain Control*, que permite ajustar el volumen, con acciones del tipo *SetLevel()* y *SetMute()*. Como cualquier otro tipo de componente, *GainControl* puede ser añadido a un *applet* o aplicación como si se trata de un componente de tipo *GUI*.

VISUALIZACIÓN DE UNA BARRA DE PROGRESO

Ya que el proceso de carga de los datos multimedia, un vídeo o una canción, consume una cierta cantidad de tiempo, el usuario tiene que esperar hasta que los datos son descargados. Por ello, un componente muy apreciado para comprobar el estado de la descarga es la barra de progreso, mediante la cual, el usuario comprueba el ritmo de descarga de los datos y se hace una idea aproximada de su finalización. Para poder añadir un componente de este estilo se utiliza la interfaz *Caching Control*, que es un tipo especial de *Control*. Se puede visualizar un componente de este estilo siguiendo las indicaciones:

En el método Init() se realizan todas las tareas de inicialización del reproductor

1. Comprobar si se dispone o está soportado ese componente mediante la llamada al método *GetControls()*.
2. Si está disponible, el objeto *Player* generará un evento *CachingControl Event* cuando la barra de progreso tenga que ser actualizada. Si el programador desea utilizar su propio componente, debe registrar este evento y actualizar su barra cuando se produce un evento *Caching ControlEvent*.

Más específicamente, cuando se utilice el componente por defecto, se deberá realizar las siguientes tareas:

1. Implementar la interfaz *Controller Listener* y mediante el método *controllerUpdate* comprobar si el evento producido es del tipo *CachingControlEvents*.
2. La primera vez que se reciba un evento *CachingControlEvents* se debería:
   2.1. Ejecutar el método *Get CachingControl* sobre el evento para crear un *CachingControl*.
   2.2. Llamar al método *GetProgress Bar* con el *CachingControl* y así conseguir la barra por defecto.
   2.3. Añadir el componente al *applet* o aplicación.
3. Cada vez que se reciba un evento *CachingControlEvent* comprobar si la descarga ha finalizado. Cuando el

método *GetContentProgress* devuelve el mismo valor que *GetContent Length*, se ha acabado, con lo que eliminamos la visualización del componente.

Aunque pueda parecer que el conjunto de tareas para llevar el control de los componentes es algo oscuro y complicado, una vez que se muestre un ejemplo el lector podrá comprobar como al familiarizarse con los pasos que se deben seguir, la tarea es sencilla y los resultados más que aceptables.

UN REPRODUCTOR CON COMPONENTES DE CONTROL

En la creación de esta nueva versión de un reproductor de vídeo vamos a tener en cuenta ciertas circunstancias que evitarían situaciones anormales, y se va a añadir una barra de progreso y un componente de control. Como se indicó anteriormente, la primera parte del desarrollo pasa por preparar todo lo necesario para la reproducción. Para ello se desarrolla el siguiente código:

```
import java.applet.*;
import java.awt.*;
import java.net.*;
import javax.media.*;

public class repro2 extends
     Applet implements
     ControllerListener {
   Player repro = null ;
   Component compvisual  = null ;
   Component compcontrol = null ;
   Component barracarga  = null ;

   public void init () {
setLayout(new BorderLayout());
     try {
           URL miurl = new URL (
     getCodeBase(), "video.mpg");
           repro =
     Manager.createPlayer( miurl
     );

     repro.addControllerListener(t
     his);
         }
     catch (MalformedURLException
     e ) {
        Fatal ( "URL no valida..
     ");
     }
     catch ( Exception e ) {
              Fatal ( "No se
     puede gestionar objeto Player
     ... ");
     }
   }
}
```

Se observa como se declaran tres objetos de tipo *Component*. El primero, **compvisual** se encargará de mostrar un panel visual donde se realizará la reproducción del vídeo. Mediante el uso de **compcontrol** se podrá manejar una interfaz donde el usuario dispone de botones de *start, stop, pause*, control de volumen y un indicador de las características de reproducción. Con **barracarga** se podrá gestionar el uso de una barra de progreso en la que el usuario observará la carga del vídeo a reproducir.

La creación de un reproductor elemental apenas cuenta con una decena de líneas de código

Se ha añadido la captura de la excepción *MalformedURLException* para realizar un control más preciso en caso de errores de carga.

El control de la reproducción se basa en los métodos *Start*, *Stop* y *Destroy*. El método *Start()* realiza una llamada al método *Start()* del objeto *Player*, por lo que el objeto pasa al estado *Realized*. A partir de ahí, se puede realizar la gestión de componentes visuales y de control. Pero no se pueden añadir directamente al *applet* ya que podrían ocurrir ciertas situaciones anormales ajenas al usuario. Se ha controlado que el objeto *Player*, **repro**, pueda ser ejecutado cuando el usuario por ejemplo, oculta el navegador y vuelve a su visualización.

El método*Stop()* se ejecutará cuando el usuario decida modificar la salida de su navegador o cuando decida ocultar el *applet* en desarrollo. De ahí que se tenga que controlar que **repro** no sea *null*. Además se ejecuta *Deallocate()*, liberando ciertos recursos que pudieran ser necesarios para otros objetos *Player*. El método *Destroy* se ejecuta cuando el *applet* está definitivamente finalizado, y la ejecución de *close()* hace que todos los recursos asignados sean liberados. En las siguientes líneas de código se muestran las versiones mejoradas de estos métodos.

```
public void start()   {
   if ( repro != null )
      repro.start();
}
public void stop() {
   if ( repro != null ) {
      repro.stop();
      repro.deallocate();
   }
}

public void destroy() {
   if ( repro != null ) {
      repro.close();
      repro = null;
   }
}
```

La gestión de los eventos como último paso contiene las novedades descritas anteriormente. Inicialmente se controla que el objeto **repro** esté aún activo, evitando así cualquier problema de sincronización. Cuando el objeto se encuentre en el estado *Realized* se pueden añadir los componentes de control y de visualización. Detectando el evento *Realize CompleteEvent* como se describió anteriormente, se añade al *applet* el componente de control y de visualización, mediante el uso de los métodos *GetControlPanelComponent* y *GetVisual Component*, como se detalla en las siguientes líneas:

```
public synchronized void
     controllerUpdate (
     ControllerEvent evt ) {
```

```
  if ( repro == null )
     return ;
  if ( evt instanceof
   RealizeCompleteEvent ) {
     // Los controles.
     if ( (compcontrol=
   repro.getControlPanelComponen
   t()) != null ) {
 add ("North",compcontrol);
     }
     // La pantalla
     if ( (compvisual=
   repro.getVisualComponent())
   != null ) {
add ("Center",compvisual);
     }
}
```

Para añadir una barra de progreso se comprueba que ocurra el evento *CachingControlEvent* y se desarrollan lo pasos comentados sobre el componente de barra de progreso. En las siguientes líneas se muestra el desarrollo.

```
// La existencia de este evento
     depende de la implementación
     del player
if ( evt instanceof
     CachingControlEvent)    {
     CachingControlEvent cce =
     (CachingControlEvent) evt;
     CachingControl       cc=
     cce.getCachingControl();
     long cc_progreso    =
     cce.getContentProgress();
     long cc_long        =
     cc.getContentLength();
     // Carga la barra al inicio
     del proceso
     if ( barracarga == null &&
       ( barracarga =
     cc.getProgressBarComponent()
     ) != null ) {
      add ( "South", barracarga );
      validate();
     }
     // Quitar la barra cuando es
     cargado
     if ( barracarga != null  &&
     cc_progreso == cc_long )
     {
 remove ( barracarga );
 barracarga = null ;
validate();
  }
}
```

Para controlar la finalización de la reproducción y su posible inicio, se comprueba el evento *EndOf Media Event* y mediante el uso del método *setMedia Time*() se sitúa la reproducción al inicio. Además se comprueba que no haya ocurrido un error grave durante la gestión de los datos que se van a reproducir mediante el evento *Controller ErrorEvent*. El método *Fatal* se encarga de mostrarnos qué tipo de problema.

```
// Vuelta el inicio y
     comenzar de nuevo
if ( evt instanceof
     EndOfMediaEvent ) {
repro.setMediaTime ( new Time
     (0));
repro.start();
}
else {
// Si ocurre un error dramatico
     if ( evt instanceof
     ControllerErrorEvent ) {
             repro = null;
        Fatal
     (((ControllerErrorEvent)evt).
     getMessage());
     }
        }
void Fatal ( String s ) {
   repro = null ;
   System.err.println ( "Error
    critico : " + s );
   throw new Error (s);
}
```

En la figura 2 se muestra el reproductor con los componentes de control. El lector ha podido comprobar que añadir nuevas posibilidades y desarrollar una aplicación más robusta sólo requiere un incremento mínimo de código. El reproductor descrito puede ser utilizado perfectamente para reproducir audio y cualquier otro tipo de vídeo como *MOV*, *AVI*, etc. En el *CD-ROM* incluido en la revista puede encontrar el código fuente.

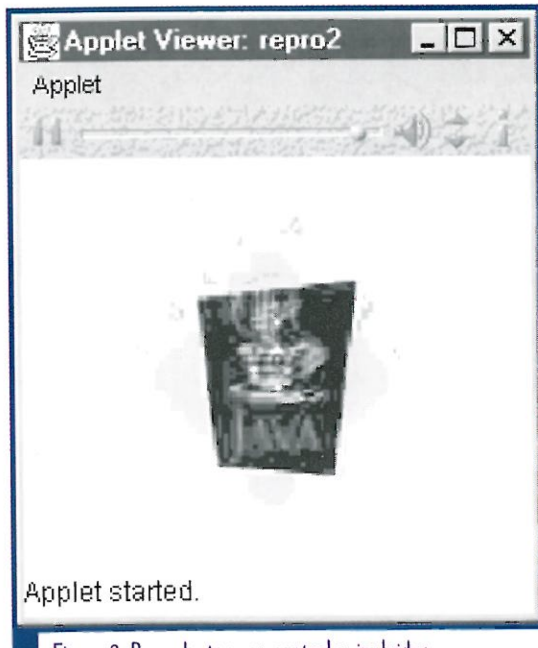


Figura 2. Reproductor con controles incluidos.

CONCLUSIÓN

El programador puede usar *Java Sound* o *Java Media Player* para crear sus aplicaciones multimedia. Gracias a las facilidades que ofrece *Java Media Player*, se pueden crear aplicaciones que reproduzcan vídeo y audio de una forma fácil y sencilla.

REFERENCIA DE LA SERIE

Multimedia con Java: sonido y vídeo

Sólo Programadores 56
INTRODUCCIÓN A JAVA MEDIA FRAMEWORK Y JAVA SOUND

Sólo Programadores 57
DESARROLLO DE APLICACIONES PARA REPRODUCIR AUDIO CON JAVA SOUND. FUNDAMENTOS DE JAVA MEDIA PLAYER

Captura de secuencias con Delphi (II)

Juan Luis Ceada (i5939@fie.us.es)

En esta entrega nos dedicaremos de lleno a ampliar las capacidades del componente *TJLCVideoPanel*, añadiéndole propiedades y métodos que nos permitan obtener uno nuevo: *TJLCVideoPanelNEW*.

INTRODUCCIÓN

En el artículo anterior desarrollamos un componente muy básico, que permitía a cualquier programador incluir vídeo en sus aplicaciones.

Además, expusimos algunos conceptos teóricos necesarios para poder comprender el proceso de implementación sin problemas.

El nuevo componente lo desarrollamos a partir del código creado en el número anterior de la revista

En esta entrega, ampliaremos el componente, añadiéndole propiedades y métodos de gran utilidad, aunque primero tendremos que profundizar en la información relativa a la *API* **avicap32.dll**, ya que el mes anterior, por problemas de espacio, no pudimos comentar todo lo necesario para el desarrollo completo del componente.

Este nuevo componente, al que denominaremos *TJLCVideoPanelNEW*, se desarrolla partiendo de cero. Aunque podríamos haber optado por usar como componente base el desarrollado en la pasada entrega, mediante herencia, hemos creído conveniente crear uno nuevo.

El motivo es muy sencillo: es necesario tener instalado *TJLCVideoPanel* para poder crear un componente del tipo *TJLCVideoPanelNEW* mediante herencia. No es muy lógico tener dos componentes que proporcionan la misma funcionalidad (y uno de ellos es mucho más potente). Eso sí, aunque no se use la herencia, se aprovecha el código anterior utilizando las tan simples operaciones de cortar y pegar.

NUEVAS PROPIEDADES Y MÉTODOS

En las tablas 1 y 2 se muestran cuáles son las nuevas propiedades y métodos que se van a incluir en el componente. La adición de nuevos elementos obliga a modificar, en parte, el código desarrollado en la entrega anterior. En primer lugar, hay que modificar el constructor del componente, de forma que inicie las nuevas propiedades.

```
function TJLCVideoPanelNEW.
     IniciarDriver(Index:Integer):
     Boolean;
var
     Retc:LongInt;
   CapParms:TCAPTUREPARMS;
begin
      Result := FALSE;
      // Nos conectamos al driver
     de captura de video
```

```
if capDriverConnect(FVentanaPre-
    view, Index) <> 0 then
begin
retc := capCaptureGetSetup(
    FVentanaPreview,
    LongInt(@CapParms),
sizeof(TCAPTUREPARMS));
if retc <> 0 then
begin
CapParms.fMCIControl   := FALSE;
CapParms.vKeyAbort    := VK_ESCAPE;
CapParms.fAbortLeftMouse:= FALSE;
CapParms.fAbortRightMouse:= FALSE;
retc := capCaptureSetSetup(
    FVentanaPreview,
    LongInt(@CapParms),
sizeof(TCAPTUREPARMS));
if retc = 0 then exit;
end;
Result := TRUE;
end
else
Raise Exception.Create('No se ha
    podido conectar el driver.');
end;
```

La función *IniciarDriver* también ha cambiado un poco. Antes simplemente conectaba la ventana de visualización al *driver*, mientras que ahora aprovechamos para establecer el valor de algunas propiedades que hacen referencia al proceso de captura. Dichas propiedades no van a ser usadas en nuestro componente, y por defecto tienen un valor que no nos conviene. Por ejemplo, por defecto el usuario puede abortar la captura mediante un clic (ya sea con el botón izquierdo o derecho del ratón). Puesto que esto no nos interesa, inmediatamente después de conectar la ventana al *driver*, obtenemos la configuración actual de la captura mediante la función *capCaptureGetSetup*. Después modificamos los campos correspondientes (por ejemplo, *fAbortLeftMouse: =FALSE*), y actualizamos la configuración mediante *capCaptureSetSetup*. Para obtener más información conviene consultar las tablas 3 y 5.

Sobre el resto de las propiedades y métodos no comentaremos nada más por ahora, puesto que el nombre y la descripción incluida en las tablas 1 y 2 resulta suficiente para comprender su utilidad.

Tabla 1. Nuevas propiedades de JLCVideoPanelNEW.

Nombre	Tipo	Acceso	Descripción
Grabando	Boolean	L	Indica si se está capturando vídeo.
Activo	Boolean	L/E	Activar/desactivar el componente.
FicheroVideo	String	L/E	Indica el nombre del AVI que capturar.
FicheroImagen	String	L/E	Indica el nombre del BMP que capturar.
FramesCaptura	Dword	L/E	FPS durante la captura.
CapturaConAudio	Boolean	L/E	Activa/desactiva la captura de audio.
HiloAparte	Boolean	L/E	Activa/desactiva la captura en segundo plano.
TiempoActivado	Boolean	L/E	Activa/desactiva la grabación de vídeo limitada por el tiempo
Segundos	Integer	L/E	Tiempo máximo de la duración del vídeo cuando TiempoActivado es true.

LA BASE DE TODO: AVICAP32.DLL

Ampliaremos ahora la información que se ofreció en la entrega anterior sobre esta *API*. Recordemos que todo el proceso de captura/visualización de vídeo se realiza mediante llamadas a esta librería. Además, no está de más volver a mencionar que para trabajar más cómodamente desde *Delphi* usaremos una unidad que encapsula la interfaz que ofrece *AVICap*. Así no tendremos que "pelearnos" con tipos de datos "raros" y punteros extraños. Se trata del fichero **AviCaptura.pas**, que podréis encontrar en el *CD-ROM* que viene con la revista.

Basándonos en la API Avicap32, ampliaremos las posibilidades del componente

En la tabla 3 se muestran las funciones incluidas en dicha unidad, que junto con las que vimos en la entrega anterior servirán para implementar el componente. Conforme vayamos viendo cómo implementar los nuevos métodos del componente, explicaremos para qué sirven y cómo se usan.

Tabla 2. Nuevos métodos de JLCVideoPanelNEW.

Nombre	Descripción
GrabarVideoDisco	Comienza la captura de vídeo. El resultado se almacena en disco duro.
GrabarVideoSinDisco	Comienza la captura de vídeo. El resulta do no se almacena en disco.
StopVideo	Detiene la captura.
GrabarImagenPortaPapeles	Captura una imagen y la almacena en el portapapeles.
GrabarImagenDisco	Captura una imagen y la almacena en disco.

Existen tres estructuras de datos, que por su importancia, merecen ser comentadas. Sus nombres son *TCapDriversCaps, TCaptureParms y TCapStatus*. Sobre la primera ya hablamos el mes pasado. Veremos ahora para qué sirven las otras dos. Puedes consultar las tablas 4 y 5, en las cuáles se muestran los campos de dichas estructuras.

La primera de las estructuras, *TCapStatus*, ofrece información sobre el proceso de captura/visualización de vídeo. Como se puede observar en la tabla indica aspectos tan útiles como que hay una captura en proceso, si estamos en modo *preview*, el número de *frames* capturados, etc. La función *CapGetStatus* devuelve una estructura de este tipo, que podemos consultar para ver cómo se está desarrollando todo.

La otra estructura, *TCaptureParms*, almacena información acerca de la configuración actual de la tarjeta capturadora. Podemos indicar (o averiguar), mediante las funciones *CapCaptureGetSetup* y *CapCaptureSetSetup* cómo queremos que se lleven a cabo las capturas. Por ejemplo, podemos hacer que la captura se lleve a cabo a un número determinado de *frames* por segundo, que incluya audio, que se pueda abortar con una simple pulsación del botón izquierdo, y que se realice en segundo plano (mediante la creación de un nuevo hilo de ejecución).

Esta estructura, y las funciones de consulta y modificación asociadas a ella, son la base de la implementación de los procedimientos *SetCapturaConAudio*, *SetFramesCaptura*, etc., que *Delphi* ejecuta cuando el usuario cambia el valor de alguna de las propiedades que proporciona el componente.

Mediante las estructuras TCapStatus y TCaptureParams podemos controlar todo el proceso de visualización/captura

Tabla 3. Algunas funciones contenidas en AviCaptura.pas.

Función	**Descripción**
CapDriverGetCaps	Devuelve las características disponibles en el hardware en una estructura de tipo TCAPDRIVERSCAPS.
CapFileSetCaptureFile	Establece el nombre del fichero por defecto que se usará al grabar vídeo en el disco duro.
CapFileGetCaptureFile	Obtiene el nombre del fichero por defecto que se usará al grabar vídeo en el disco duro.
CapFileSaveDIB	Almacena el contenido del frame actual en un ficheroDIB (con extensión .BMP)
CapEditCopy	Copia el contenido del frame actual al portapapeles.
CapCaptureStop	Detiene la captura de vídeo.
CapGrabFrameNoStop	Captura una única imagen, sin importar el modo en el que se esté. Si es necesario, pasará de overlay a preview automáticamente.
CapCaptureSequence	Comienza la captura de vídeo.
CapCaptureAbort	Aborta la operación de captura en proceso.
CapGetStatus	Devuelve una estructura de tipo TCapStatus.
CapCaptureGetSetup	Devuelve una estructura de tipo TCaptureParms.
CapCaptureSetSetup	Cambia los parámetros de captura/ visualización de vídeo.
CapSetCallBackOnYield CapSetCallbackOnVideoStream CapSetCallbackOnError CapSetCallbackOnStatus CapSetCallbackOnFrame CapSetCallbackOnWaveStream	Se encargan de instalar las funciones de callback que se ejecutarán cuando se produzca un suceso determinado. Ej: CapSetCallbackOnFrame (Ventana,MiFuncion) hará que se ejecute la función MiFuncion cada vez que llegue un nuevo frame. CapSetCallBackOnFrame (Ventana, 0) la desactiva.

Sólo una cosa más al respecto. Como se puede apreciar, en la implementación de nuestro componente no se han usado todos los campos que proporcionan estas estructuras. Nosotros sólo usaremos los más importantes, y que consideramos de interés general. Para aplicaciones más especificas es posible que sea necesario utilizar algunos más.

Por ejemplo, en algunos casos podría ser útil que apareciese un cuadro de diálogo que obligase al usuario a pulsar *OK* para que diese comienzo la captura. Para ello, bastaría con añadir una nueva propiedad, llamada por ejemplo *UsuarioPulsaOK*, en cuyo procedimiento de asignación *SetUsuarioPulsaOK* accediese al campo *fMakeUserHitOKToCapture* de la estructura, pusiese su valor a *true*, y enviase dicha estructura a la tarjeta.

Tal y como se puede observar, se trata de un proceso bastante simple y sencillo, al alcance de cualquier programador con unos mínimos conocimientos de *Delphi*.

IMPLEMENTANDO EL NUEVO COMPONENTE

Normalmente, cuando se desarrolla un componente se comienza por declarar su interfaz, así que eso es precisamente lo primero que haremos. Usaremos como base el código creado en la entrega anterior, al que añadiremos las nuevas propiedades, métodos y eventos. El proceso es fácil: en la sección *private*, añadiremos las variables internas necesarias para almacenar el valor de las nuevas propiedades, así como la declaración de los procedimientos de asignación/lectura de dichas propiedades y eventos.

Ampliar el componente añadiendo nuevas propiedades es un proceso muy sencillo

En la sección *public* declararemos los procedimientos/funciones que estarán disponibles para el usuario del componente. Como hemos dicho, es un proceso fácil, pero tedioso. Afortunadamente, podemos recurrir al código fuente, incluido en el *CD-ROM* de la revista, lo que ahorrará algunos minutos de trabajo. Acude a él si no tienes muy claro cómo se declara la interfaz de un componente.

LAS NUEVAS PROPIEDADES

Cuando se asigna un nuevo valor a determinadas propiedades, se ejecuta el procedimiento *SetNombrePropiedad*. Dicho procedimiento posee un parámetro de entrada, que es exactamente del mismo tipo que la propiedad a la que se pretende asignar el valor. Internamente, se asigna dicho parámetro a la variable privada que almacena el valor de la propiedad (recordemos que dicha variable recibe el nombre de la propiedad, precedida por la letra F. Ej: propiedad **HiloAparte**: variable **FHiloAparte**). Un ejemplo de procedimiento de asignación aparece en el siguiente código:

```
procedure TJLCVideoPanel.
     SetHiloAparte(Valor:Boolean);
var
  retc:LongInt;
  CapParms:TCAPTUREPARMS;
begin
 FHiloAparte:=Valor;
 if FVentanaPreview=0 then exit;
 retc := capCaptureGetSetup(
     FVentanaPreview,
     LongInt(@CapParms),
      sizeof(TCAPTUREPARMS));
 if retc <> 0 then
 begin
```

Tabla 4. Estructura TCapStatus. Algunas funciones devuelven una estructura de este tipo, que almacena el estado actual de la captura.

Campo	Descripción
UIImageWidth	Ancho del frame capturado.
UIImageHeight	Alto del frame capturado.
FLiveWindow	¿Estamos en modo preview?
FScale	¿Activado el reajuste de tamaño?
FUsingDefaultPalette	¿Usando la paleta de colores por defecto?
FAudioHardware	¿Existe hardware para la captura de audio?
FCapFileExists	¿Existe el fichero de captura?
DwCurrentVideoFrame	Número de frames capturados.
DwCurrentVideoFramesDropped	Frames que no se han podido capturar.
DwCurrentWaveSamples	Número de samples de sonido capturados.
DwCurrentTimeElapsedMS	Tiempo de captura transcurrido.
HPalCurrent	Puntero a la paleta actual.
FCapturingNow	¿Estamos capturando vídeo?
WNumVideoAllocated	Buffers vídeo creados.
WNumAudioAllocated	Buffers de audio creados.

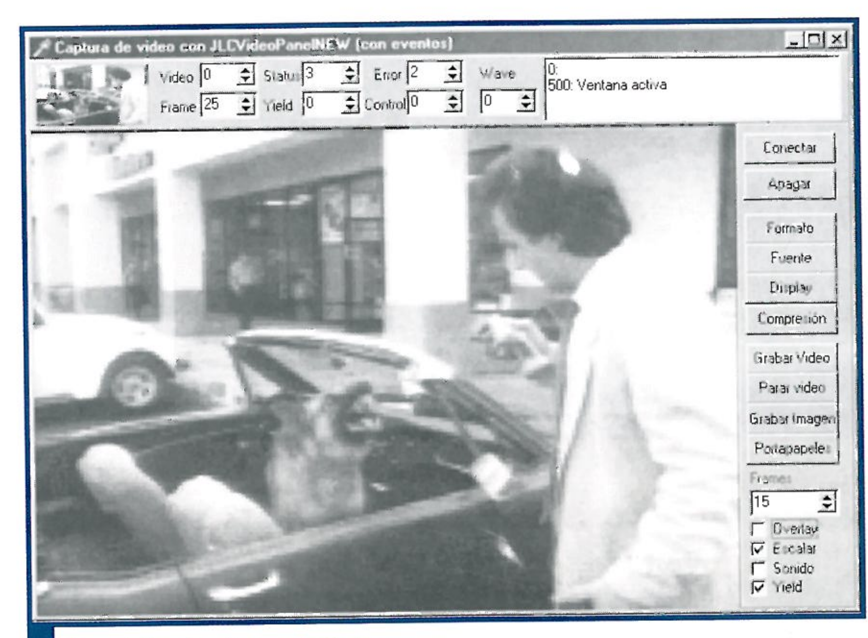


Figura 1. Visualizando en modo Preview.

Tabla 5. Estructura TCaptureParms. Se han omitido algunos campos para el control de dispositivos MCI.

Campo	Descripción
DwRequestMicroSecPerFrame	FPS. cuando se captura. Valor en microsegundos. Por ejemplo, un valor de 66666ms equivale a capturar 15 imágenes por segundo.
FMakeUserHitOKToCapture	Si vale True aparece un cuadro de diálogo antes de comenzar la captura. Cuando el usuario pulsa Aceptar, da comienzo la captura.
WPercentDropForError	Porcentaje máximo de pérdida de imágenes capturadas. Por defecto es el 10% del total.
FYield	¿La captura se realiza en segundo plano?
DwIndexSize	Número de entradas en un fichero AVI. Cada frame usa una entrada, por lo que, en cierto modo, este parámetro controla el tamaño de la captura. Por defecto, 32K entradas (unos 15 minutos a 30 FPS).
FCaptureAudio	¿Capturar audio junto con el vídeo?
WChunkGranularity	Tamaño de un bloque lógico en el fichero AVI. Por defecto se usa el tamaño de cluster actual.
FusingDOSMemory	No se usa en aplicaciones Win32.
WnumVideoRequested	Máximo número de buffers a asignar para vídeo.
WnumAudioRequested	Máximo número de buffers a asignar para audio.
VkeyAbort	Tecla que aborta la operación de captura. Por defecto, la tecla Esc. Es posible cambiarla mediante la función RegisterHotkey.
FabortLeftMouse	¿Abortar al pulsar el botón izquierdo del ratón?
FabortRightMouse	¿Abortar al pulsar el botón derecho del ratón?
FlimitEnabled	¿Limitar el tiempo de grabación?
WtimeLimit	Duración máxima de la captura, en segundos. FlimitEnabled debe estar a True.

```
  CapParms.fYield:= Valor;
retc := capCaptureSetSetup
     (FVentanaPreview,
     LongInt(@CapParms),
     sizeof(TCAPTUREPARMS));
 end;
end;
function TJLCVideoPanelNEW.Get
     Grabando:boolean;
begin
   Result:=FGrabando;
end;
```

Este procedimiento es el que se ejecuta cuando se cambia el valor de la propiedad *HiloAparte*. Como se puede observar, lo primero que se hace es asignar el contenido del parámetro **valor** a la variable *FHiloAparte*. A continuación, obtenemos en la variable local *CapParams* los parámetros actuales de la captura de vídeo. Modificamos el campo deseado (en este caso, *FYield*), y volvemos a enviar la estructura *CapParams* a la tarjeta. Esto mismo habrá que realizarlo con las diferentes propiedades del componente que afectan directamente a la captura de vídeo (como *FramesCaptura*, *CapturaConAudio*, etc.) Básicamente es el mismo código, sólo cambia el campo de la estructura *CapParams* que hay que modificar.

El único procedimiento que se sale de la norma es *SetEscalaPreview*, en el que no es necesario solicitar ningún tipo de estructura de datos. Sólo hay que usar la función *capPreviewScale* pasándole el parámetro adecuado.

Todos los procedimientos de asignación comparten el mismo código, sólo cambia el campo de la estructura CapParams

Hay un detalle más a tener en cuenta. Normalmente cuando se desarrolla un componente, si en tiempo de diseño mediante el inspector de objetos, modificamos alguna de sus propiedades, inmediatamente se ejecuta el procedimiento de asignación correspondiente, y el cambio de valor es comunicado al componente. En caso contrario, aun a pesar de modificar las propiedades no se producirá el efecto deseado. Un ejemplo: si en tiempo de diseño cambia la propiedad *HiloAparte* de **true** a **false**, y en tiempo de ejecución, nada más activar el componente, comprueba cuál es su valor, obtendrá como resultado el valor **false**, ¡aunque internamente, la tarjeta funcionará como si estuviese a **true**!. El cambio no se ha visto reflejado en el componente, ya que cuando se realizó, éste no estaba activo.

Básicamente, modificamos el código del procedimiento *Conectar*, para que, inmediatamente después de crear la ventana de visualización (pero antes de comenzar a mostrar imágenes), se llame "manualmente" a cada

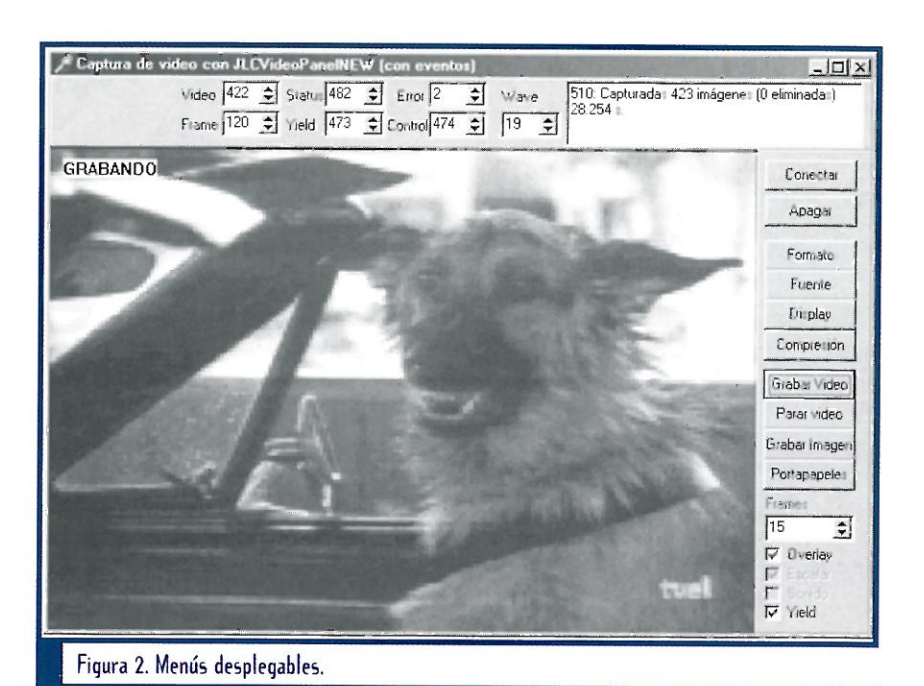


Figura 2. Menús desplegables.

uno de los procedimientos de asignación, pasándole como parámetro la variable que contiene el valor actual de la propiedad. Así nos aseguramos que todo se inicia correctamente.

```
procedure TJLCVideoPanelNEW.Conectar;
begin
  if AbrirDriver then
  begin
SetEscalaPreview(FEscalaPreview);
SetFramesCaptura(FFramesCaptura);
SetCapturaConAudio(FCapturaCon
    Audio);
SetEscalaPreview(FEscalaPreview);
SetFramesCaptura(FFramesCaptura);
SetHiloAparte(FHiloAparte);
SetTiempoActivado(FTiempoActivado);
SetSegundos(FSegundos);
MostrarVideo;
InstalarCallBack;
end
else
raise Exception.Create('No se ha
    podido conectar el video');
end;
```

Además de los procedimientos de asignación, existe uno que se ejecuta cada vez que el usuario consulta el valor de la propiedad *Grabando*. Dicha función accede a la variable *FGrabando* y devuelve su valor. La variable *FGrabando* es actualizada por una función *callback*, que posteriormente asociaremos a un evento. Puesto que los eventos serán explicados en la próxima entrega, habrá que tomarse esto como un dogma de fe, hasta el próximo artículo, en donde será explicado convenientemente.

LOS NUEVOS MÉTODOS

Los métodos que vamos a poner a disposición del usuario aparecen en la tabla 2. Veamos más detenidamente cada uno de ellos.

- **GrabarVideoDisco, GrabarVideoSinDisco, StopVideo.** El primero de los métodos se encarga de poner en marcha el proceso de captura de vídeo. En primer lugar se pasa a modo *preview*, después se asigna un nombre al fichero *AVI*. Dicho nombre de fichero lo toma de la propiedad *FicheroVideo*, la cual, por defecto tiene como valor la cadena **video.avi**. A continuación da comienzo la captura de vídeo (*CapCaptureSequence*).

El siguiente método, *GrabarVideoSinDisco*, permite iniciar una captura, pero los *frames* capturados no se almacenan en ningún sitio. Simplemente se capturan.

Cada vez que se captura un *frame*, se produce un evento, que el usuario puede usar para procesar las imágenes capturadas.

No olvide incluir los ficheros MMSystem y Avicaptura en la sección Uses de sus aplicaciones

Por último, *StopVideo* finaliza la captura de vídeo en proceso (suponiendo que haya alguna).

- **GrabarImagenPortaPapeles, GrabarImagenDisco.** El primer método graba una imagen en soporte magnético, en formato *DIB (Device Independent Bitmap)*. El nombre del fichero lo obtiene de la propiedad *FicheroImagen*, siendo por defecto **imagen.bmp**. Para ello, en primer lugar se llama a la función *CapGrabFrameNoStop*, que, por sí sola, pasa a modo *preview*, captura la imagen, y vuelve a modo *overlay*. Después, una llamada a *CapFileSaveDIB* resuelve el problema del almacenamiento en disco.

El siguiente método es similar, sólo que en vez de copiar el *frame* a un archivo del disco, se copia al portapapeles, usando para ello la función *CapEditCopy*..

INSTALACIÓN Y PRUEBAS DEL COMPONENTE

Si el lector trabaja asiduamente con *Delphi*, es seguro que sabe instalar componentes, pero en cualquier caso junto a los fuentes se ha incluido un fichero **leeme.txt** donde se explican todos los pasos que dar para conseguir-

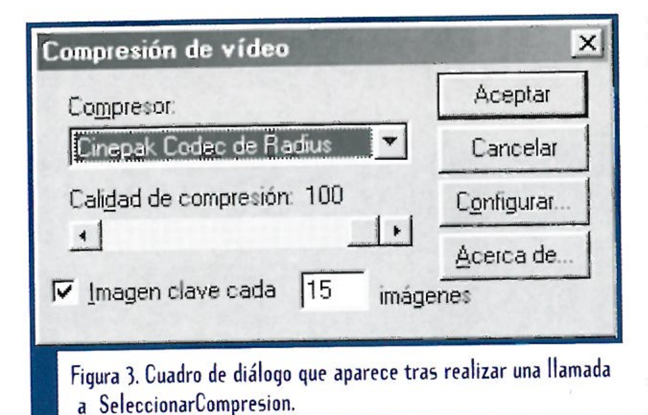


Figura 3. Cuadro de diálogo que aparece tras realizar una llamada a SeleccionarCompresion.

lo. No hay ningún problema por tener instalado el componente desarrollado el mes anterior, ya que ambos pueden convivir juntos.

La aplicación es una mejora de la desarrollada el mes anterior. Hay varias novedades. En primer lugar, es necesario incluir las librerías *MMSystem* y *AviCaptura* en la sección *uses*. Esto nos permite tener acceso a algunos tipos de datos y constantes. A primera vista, se puede apreciar que existen más elementos en la aplicación de los que se describen a continuación.

Esto es debido a que dichos controles están relacionados con los eventos que dispone el componente. Por problemas de espacio, no podemos explicar cuándo se produce cada evento y qué hace el código asociado a cada uno de ellos, cosa que dejamos para la próxima entrega.

Si sientes curiosidad, puedes ir intentando averiguar qué es lo que se hace en respuesta a cada uno de los eventos, hasta que el próximo mes resolvamos todas las dudas.

En el lado derecho, aparecen nuevos botones y *checkboxes*. Tenemos un botón para iniciar la captura de vídeo, otro para detenerla y uno más para grabar una imagen en formato *BMP* en disco o en el portapapeles. El control **Frames** controla el número de *FPS* que usar durante la visualización/captura en modo *preview*. Es posible controlar estos parámetros de forma independiente, pero en este caso hemos preferido unificarlos.

Los *checkbox Sonido* y *Yield* permiten indicar, respectivamente, si el vídeo capturado incluirá audio y si la captura se realiza en segundo plano (cuidado con esto, si se desactiva puede dejar la aplicación aparentemente bloqueada). Los otros *checkboxes*, *Overlay* y *Escalar*, permiten pasar del modo *overlay* al *preview* y viceversa, y activar el redimensionado automático de la imagen en modo *preview*, respectivamente.

Conviene hacer pruebas, para ver qué método de compresión se adapta a nuestras necesidades

Sólo nos queda comentar algo acerca del botón **Compresión**. Su pulsación hará que aparezca una pantalla similar a la de la figura 3. Permite elegir y configurar el sistema de compresión que usar durante la captura. El número de sistemas de compresión depende del *hardware* y *software* que tengamos ya instalado en nuestro equipo.

Cada uno tiene sus propias características, por lo que es conveniente realizar diversas pruebas, hasta que demos con el que mejor se ajuste a nuestros propósitos.

En el *CD-ROM* podréis encontrar varios vídeos, grabados usando diferentes formatos de compresión, para que aquellos que no dispongan de tarjeta sintonizadora puedan tener una idea de la calidad y *ratio* de compresión de cada método. Cuidado: la grabación de vídeo con compresión simultanea es un auténtico devorador de recursos del sistema (sólo recomendado para P-200 o superiores).

CONCLUSIÓN

Con las propiedades y métodos que se han implementado debería ser suficiente para satisfacer las necesidades de la gran mayoría de los usuarios/programadores, pero es posible ampliarlo bastante.

Como se comentó en el articulo anterior, la *API avicap32.dll* contiene bastantes más funciones de las que aquí se usan. Sólo es cuestión de estudiar un poco el fichero *AviCaptura.pas*, y a partir de ahí, ampliar el componente tanto como se desee. Por supuesto, podrás encontrar información sobre el tema en el propio *Delphi* (fichero *mm.hlp* o *mmedia.hlp*), así como en *Internet*.

Puedes probar en la dirección **www.microsoft.com**, o introducir en cualquier buscador las frases *AVICAP32*, *"video capture"* o similares. En **www.clubdelphi.com** podrás encontrar actualizaciones y nuevas versiones del componente.

En el próximo artículo, seguiremos explicando cómo funciona el componente (en concreto la parte relacionada con los eventos).

REFERENCIA DE LA SERIE

Captura de secuencias con Delphi

Sólo Programadores 57
REALIZAR UN COMPONENTE PARA CAPTURAR Y VISUALIZAR VÍDEO

Desarrollo de aplicaciones con videoconferencia (IV)

Constantino Sánchez Ballesteros (constantino@nexo.es)

Mediante el *ILS (Internet Locator Server)* podemos localizar a usuarios determinados para poder establecer una conferencia. Sin éste no podríamos determinar a quién queremos llamar, ni por tanto, comenzar una comunicación.

ATRIBUTOS EXTENDIDOS DEL ILS

Internet Locator Server (ILS) permite a los usuarios de un programa de conferencia localizar a otros usuarios que estén ejecutando el mismo *software* y conectarse entre ellos. Para aparecer en una lista de usuarios, cada persona debe registrarse en el *ILS*. El *ILS* rastrea cada persona registrada como un objeto *Usuario*. Cada objeto *Usuario* tiene diversos atributos estándar, incluyendo el nombre, apellidos, dirección *e-mail* y ciudad. Además, una aplicación puede crear atributos extendidos para sus usuarios y crear informes de esos atributos. Hay dos formas de establecer y obtener atributos extendidos:

- Crear el usuario sobre un cliente a través de las *APIs ILS* incluidas en *NetMeeting SDK*.
- Crear el usuario y los atributos a través de los objetos del servidor incluidos en *Microsoft ILS Server 1.0*.

LIMITACIONES

Mediante *ILS Server 1.0* todos los nombres de los atributos extendidos deben ser cadenas numéricas de texto de tipo *BSTR*.

Esto significa que no podemos crear un atributo nombrado como **Nombre del jugador**, por ello, debemos crear un atributo numerado (por ejemplo, **401** o **535**).

Los atributos extendidos deberían ser números mayores de **400**. En la tabla del siguiente apartado se enumeran los atributos de nombres actualmente utilizados por *NetMeeting*. Los atributos extendidos son específicos para cada aplicación, por lo tanto, el atributo **450** de una aplicación no colisionará con el **450** de otra aplicación diferente. Por esta razón, todas las búsquedas basadas en atributos extendidos deben incluir el identificador de la aplicación. *NetMeeting* utiliza el identificador de aplicación *ms-netmeeting*.

Es posible crear atributos extendidos, definidos por el usuario, para reflejar las características de éste en la conferencia

Si nuestra aplicación usa este identificador de aplicación, debemos elegir atributos extendidos que no colisionen con aquellos utilizados por *NetMeeting*.

EL CLIENTE

NetMeeting utiliza atributos extendidos para indicar que un usuario de *ILS* está en una llamada, cuándo tiene o no capacidades de audio/vídeo y la categoría a la que pertenece el usuario. La interfaz de usuario de *NetMeeting* también soporta búsqueda de usuarios basadas en esos atributos.

ESTABLECIENDO ATRIBUTOS

Los atributos extendidos se activan a través del método *IIlsUser::SetExtendedAttribute*, que toma dos parámetros - nombre del atributo extendido y valor, ambos son del tipo *BSTR*. *ILS* trata los nombres de atributos como etiquetas de propiedad *MAPI* y utiliza la macro *PROP_TAG* para codificar el nombre antes de establecer el atributo. Debemos crear la misma codificación cuando obtenemos un atributo extendido de *NetMeeting* para que los nombres se correspondan. Por ejemplo, para hacer que el estado de usuario sea de "actualmente en una llamada " utilizaríamos el siguiente código:

```
DWORD dwAtt =
     PROP_TAG(PT_STRING8, 400);
wsprintf(szName, TEXT("%lu"),
     dwAtt);
LPTSTR_to_BSTR( szName, bstrName);
LPTSTR_to_BSTR( "1", bstrValue);
     // asignamos al usuario como
     "en una llamada"
IIlsUser::SetExtendedAttribute
     (bstrName, bstrValue);
IIlsUser::Update()
```

Nota: *La implementación de la macro LPTSTR_to_BSTR se ha dejado como ejercicio para el lector.*

OBTENCIÓN DE ATRIBUTOS

La llamada a *IILsUser::GetExtendedAttribute* o *IILsUser::GetAllExtendedAttributes* no provoca una reacción del servidor. Todos los nombres de usuario deseados deben ser especificados cuando el objeto *Usuario* se obtenga del servidor. Las llamadas posteriores a *GetExtendedAttribute* o *GetAllExtendedAttributes* devolverán los valores de los atributos.

El ILS rastrea cada persona registrada como un objeto Usuario

Para obtener un objeto *Usuario* con atributos extendidos debemos crear un objeto *Atributo* con los nombres propios y pasar este objeto a los métodos *GetUser* o *EnumUsers*, tal y como se muestra en el siguiente extracto de código:

```
LPTSTR_to_BSTR(&bstrEmpty, "");
// En una llamada
wsprintf(szProperty, TEXT("%lu"),
     PROP_TAG(PT_STRING8, 400));
LPTSTR_to_BSTR(&bstrName, szProperty);
hr = pAttrib->SetAttribute(bstrName, bstrEmpty);
// Tipo de usuario
wsprintf(szProperty, TEXT("%lu"),
     PROP_TAG(PT_STRING8, 600));
LPTSTR_to_BSTR(&bstrName, szProperty);
hr = pAttrib->SetAttribute(bstrName, bstrEmpty);
// Ahora que el objeto atributo
     se ha configurado,
// obtenemos los objetos Usuario
hr = pIls->EnumUsers (pServer,
     pFilter, pAttrib, NULL, &uReqID);
IIlsMain::EnumUsers
```

Cuadro 1. Diagrama conceptual de un objetoILS.

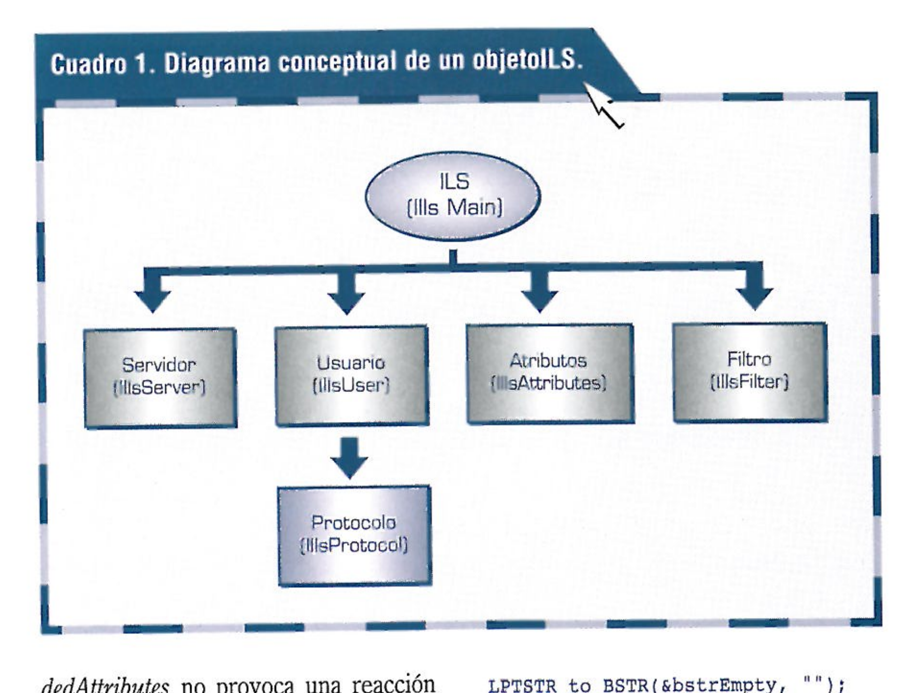


Tabla 1. Atributos de NetMeeting.

Atributo	Descripción	Posibles valores	"nombre"
400	Indicador de llamada	"0","1"	
501	Capacidad para enviar audio	"0","1"	
503	Capacidad para enviar vídeo	"0","1"	
600	Categoría del usuario	"1" =	Personal
		"2" =	Negocios
		"4" =	Adultos

SÓLO PROGRAMADORES

Tabla 2. Método EnumUsers .

Enumera la lista de objetos Usuario presentando los usuarios actualmente registrados en el servidor ILS.

- Devuelve uno de los siguientes valores:

S_OK	Operación realizada.
ILS_E_NOT_IMPL	Puntero al enumerador de usuarios no es NULL; la interfaz síncrona no está aún implementada.
ILS_E_POINTER	Dirección del objeto servidor, filtro o identificador es NULL.
ILS_E_NOT_INITIALIZED	IIlsMain interfaz no se ha inicializado.
ILS_E_MEMORY	No se pudo localizar memoria suficiente.

- pServer : Objeto servidor ILS que especifica el servidor que se pretende enumerar.
- pFilter: Objeto Filtro. Si este parámetro es NULL la enumeración se crea sin ningún filtro.
- pAttrib: Atributos del objeto que contienen los nombres de los atributos extendidos obtenidos del servidor. Si este parámetro es NULL, la enumeración se creará sin ningún tipo de atributo extendido.
- ppEnumUsers: Si el método se llama de forma síncrona, un puntero recibirá el objeto Enumerador. Actualmente se debe establecer este valor como NULL.
- puReqID: Dirección de un buffer de tipo ULONG para recibir el identificador requerido.

```
HRESULT IIlsMain::EnumUsers(
     IIlsServer   *pServer,
     IIlsFilter   *pFilter,
     IIlsAttributes  *pAttrib,
     IEnumIlsUser   **ppEnumUsers,
     ULONG   *puReqID);
```

Actualmente, este método sólo crea operaciones asíncronas, pero la interfaz se ha definido para permitir futuras implementaciones de operaciones síncronas.

Cuando es llamada de forma asíncrona, la aplicación recibe la notificación *IlsNotify::EnumUsersResult* para el resultado de la enumeración. Después de que el objeto *Usuario* junto con los atributos extendidos especificados se ha obtenido del servidor, la obtención del valor de un atributo es fácil mediante la llamada al método *GetExtendedAttribute*, tal y como se muestra en el siguiente código:

```
DWORD dwAtt =
     PROP_TAG(PT_STRING8, 501);
wsprintf(szName, TEXT("%lu"),
     dwAtt);
LPTSTR_to_BSTR( szName, bstrName);
pUser->GetExtendedAttribute (bstrName, &bstrValue);
```

Como se dijo anteriormente, el objeto *Usuario* debe haber sido obtenido del servidor con el nombre de atributo especificado. Los valores de múltiples atributos pueden devolverse mediante la creación de un objeto atributo con los nombres deseados y llamando a *GetAllExtendedAttributes*.

En un servidor ILS se albergan todos los datos de los usuarios que intervienen en una conferencia

Cuando la llamada a *GetAllExtendedAttributes* retorna, el objeto *Atributo* contendrá los valores correctos para los atributos nombrados. Una vez más, el objeto *Usuario* debería haberse obtenido del servidor con los atributos extendidos especificados.

```
LPTSTR_to_BSTR(&bstrEmpty, "");

// En una llamada
wsprintf(szProperty, TEXT("%lu"),
     PROP_TAG(PT_STRING8, 400));

LPTSTR_to_BSTR(&bstrName, szProperty);
hr = pAttrib->SetAttribute(bstrName, bstrEmpty);

// Tipo de usuario
wsprintf(szProperty, TEXT("%lu"),
     PROP_TAG(PT_STRING8, 600));

LPTSTR_to_BSTR(&bstrName, szProperty);
hr = pAttrib->SetAttribute(bstrName, bstrEmpty);

//... Gestionamos otros atributos
     aquí...
hr = pUser->GetAllExtendedAttributes(&pAttrib);
```

EL SERVIDOR

Este apartado se presenta a los autores de páginas *ASP (Active Server Pages)* activadas con *ILS* para utilizarlas con aplicaciones desarrolladas usando las interfaces provistas en el *SDK de*

NetMeeting 2.1. Podemos crear usuarios sobre el servidor utilizando objetos *ASP*. Normalmente, para acceder a los atributos extendidos utilizando páginas *ASP*, un autor de *Web's* puede especificar el atributo preguntando dicho atributo nombrado como *ilstNNNN*, donde *NNNN* es el número de atributo. De cualquier modo, debido a los problemas del *ILS Server 1.0*, si queremos acceder a un atributo extendido que nuestra aplicación ha asignado a través de la *API ILS*, necesitaremos agregar el valor **0x8000** (32768 decimal) al número de atributo y establecer un prefijo que contenga la cadena de texto *ilst*. Por ejemplo, para acceder a un atributo nombrado como 601, deberíamos referirnos al atributo *ilst33369* en el código *ASP*. Esto sólo atañe a los casos donde utilicemos *ASP* para acceder a los atributos extendidos que fueron creados a través de la *API* cliente *ILS*. Si estamos creando y preguntando por atributos a través de *ASP*, no necesitaremos realizar el paso anterior.

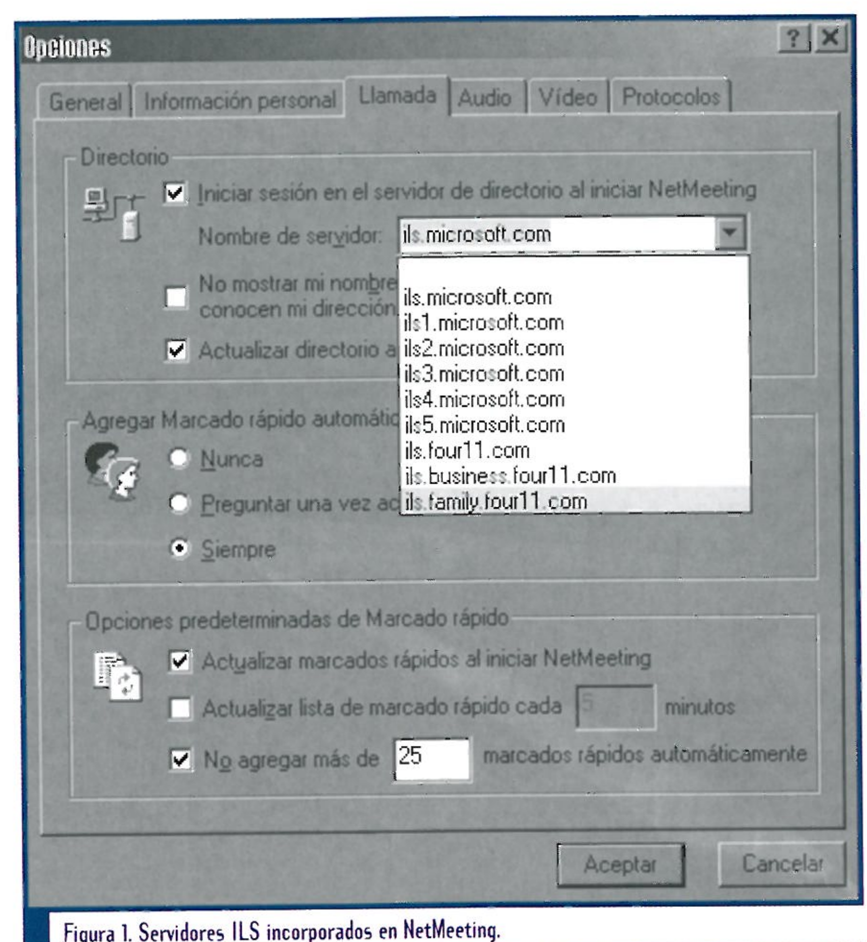


Figura 1. Servidores ILS incorporados en NetMeeting.

Los filtros nos ayudan principalmente a efectuar selecciones de determinados usuarios

El siguiente código asume que un objeto *Servidor ILS* llamado **ils** ha sido creado acorde con las instrucciones de la documentación sobre *ILS* y un objeto *Usuario* llamado **testuser** existe:

```
' Define los nombres de atributos
    extendidos aquí...
strAudio = "ilst33269"
    ' 501
strVideo = "ilst33271"
    ' 503
strIncall = "ilst33168"
    ' 400
strUserType = "ilst33368"
    ' 600
' Establece los nombres para los
    atributos extendidos
UserType = 2  Usuario de negocios
Audio = 1     Capacidad de Audio
InCall = 0   No se encuentra en
    una llamada
Video = 1     Capacidad de Vídeo
App = "ms-netmeeting"    ' Nombre
    de la aplicación
' Construye la cadena de los
    valores para modificar la
    aplicación.
' modop=0 porque los atributos
    extendidos son específicos de
    la aplicación
NMInfo = "cna=testuser" & _
    "&modop=0" & _
    "&obj=rtperson" & _
    "&appid=" & App & _
    "&appmime=text/iuls" & _
    "&protid=h323" & _
    "&protmime=text/h323" & _
    "&" & strVideo & "=" & Video
    & _
    "&" & strIncall & "=" &
    Incall & _
    "&" & strAudio & "=" & Audio
    & _
    "&" & strUserType & "=" &
    UserType
' Llamada actual para modificar
    el usuario
modifyperson = ils.modify(NMInfo)
```

La obtención de usuarios basados en un atributo extendido es relativamente fácil. Simplemente buscamos por todos los *IDs* de los usuarios donde el atributo especificado tenga el valor correcto.

Para encontrar todos los usuarios del *ILS* que son listados como usuarios de negocios, utilizaremos la siguiente búsqueda:

```
' Construimos la cadena para
    especificar el criterio de
    búsqueda.
```

Tabla 3. Método CreateFilter.

Crea un objeto Filtro que pueda ser utilizado o manipulado.

- Devuelve uno de los siguientes valores:

S_OK	Operación realizada.
ILS_S_SERVER_MAY_NOT_SUPPORT	Servidor LDAP puede que no soporte esos tipos de operaciones de filtros.
ILS_E_POINTER	ppFilter es NULL.
ILS_E_PARAMETER	La operación Filtro y tipo de Filtro son incompatibles.
ILS_E_FILTER_TYPE	Tipo de Filtro no válido.
ILS_E_MEMORY	Imposible localizar memoria requerida para realizar la operación.

- FilterType: Tipo de Filtro, el cual puede ser simple (ILS_FILTERTYPE_SIMPLE) o compuesto (ILS_FILTERTYPE_COMPOSITE).
- FilterOp: Operador del objeto Filtro. Están disponibles las siguientes opciones:
- FilterOp debe ser uno de los siguientes valores si FilterType corresponde al valor ILS_FILTERTYPE_SIMPLE.

  ILS_FILTEROP_EQUAL
  ILS_FILTEROP_APPROX
  ILS_FILTEROP_LESS_THAN
  ILS_FILTEROP_GREATER_THAN
  ILS_FILTEROP_EXIST

FilterOp debe ser uno de los siguientes valores si FilterType es ILS_FILTERTYPE_COMPOSITE.

  ILS_FILTEROP_AND
  ILS_FILTEROP_OR
  ILS_FILTEROP_NOT

- ppFilter: Dirección donde está ubicado el puntero del objeto Filtro.

```
fndstr = "obj=rtperson" & "&max-
    req=0" & _
    "&appid=" & App & "&cna=*" &
    _
    "&" & strUserType & "=" &
    UserType

' Llamada actual para encontrar
    los usuarios

ils.find(fndstr)
```

IMPLEMENTACIÓN DE FILTROS EN CONSULTAS ILS

Dado que la popularidad y utilización de los servidores *ILS* incrementa notablemente, será muy importante limitar la cantidad de información devuelta de una búsqueda del servidor *ILS*. Las interfaces *COM* del *ILS* soportan el uso de filtros para restringir la cantidad de datos devueltos por una consulta. Este tutorial demuestra el procedimiento para construir un filtro común que permitirá a nuestra aplicación localizar rápidamente un rango de usuarios del servidor.

LIMITACIONES

Tanto los atributos estándar como los extendidos pueden utilizarse en los filtros. Los atributos deben referenciarse por su valor de índice (no por sus nombres de texto). Los índices de los atributos estándar pueden encontrarse en las definiciones *ILS_STDATTR_** dentro del archivo **Ils.idl**. Los índices de los atributos extendidos deben ser definidos por el usuario.

Las interfaces COM del ILS soportan el uso de filtros para restringir la cantidad de datos devueltos

Para efectuar los filtros se puede utilizar en los valores de los atributos el carácter * para indicar que debería corresponder a cualquier cadena. De todas formas, la versión *1.0* de *ILS Server* no soporta correspondencia en subcadenas. Versiones superiores permiten búsquedas de cadenas parciales. Por ejemplo, un filtro creado para la cadena "**($1=*)&($5=san*)**" retornará todos los identificadores de usuarios cuyos apellidos comiencen con "**san**".

OBJETOS FILTRO

Un filtro puede ser tanto simple como compuesto. Un filtro simple es aquél que relata directamente un nombre de atributo y su valor de atributo asignado. Un filtro compuesto consiste

en dos o más filtros simples u otros filtros compuestos.

Un filtro compuesto consiste en dos o más simples u otros de ellos compuestos

Por ejemplo, el siguiente filtro (creado en pseudo-código) encontrará todas las personas que estén viviendo en **Salamanca**, y ejecutando *NetMeeting* o cualquier otra aplicación basada en *H323*. Además, esas personas tendrán el último apellido **"San"** pero no tendrán el primer nombre **"José"**. En este ejemplo, el filtro compuesto **A** consiste de cuatro subfiltros: **B, C, F,** y **H**.

```
Composite filter A (AND) {
    Simple filter B
    (lastname=San)
    Composite filter C (OR) {
        Simple filter D (appna-
    me=ms-netmeeting)
   Simple filter E (protid=h323)}
    Composite filter F (NOT) {
 Simple filter G (firstname=José)
    }
Simple filter H (city=Salamanca)}
```

La representación de este filtro se basa en el diseño de filtros *LDAP*. De todos modos, la interfaz de filtros *ILS* no puede aceptar un filtro de cadena *LDAP* por dos razones:

1. La aplicación no conoce los nombres de los atributos predefinidos en el objeto *Usuario* y objeto *Protocolo*.
2. La aplicación no tiene el conocimiento de la representación (cliente y/o servidor) de valores de atributos de tipo no texto.

La cadena correspondiente del filtro *LDAP* se parecería a:

```
(&(userid=*)(&(lastname=San)
(|(appid=ms-
    netmeeting)(protid=h323))
(!(firstname=José))(city=Salaman-
    ca))
```

Tabla 4. Método StringToFilter.

```
IIlsMain::StringToFilter
HRESULT IIlsMain::StringToFilter(
     BSTR  bstrFilterString,
     IIlsFilter  **ppFilter);
```

Crea un objeto Filtro que puede ser utilizado o manipulado supliendo una descripción sintáctica del filtro deseado.

- Devuelve uno de los siguientes valores:

S_OK	Operación realizada.
ILS_E_POINTER	La cadena del Filtro es NULL.
ILS_E_FAIL	No se pudo convertir la cadena.
ILS_E_MEMORY	Imposible localizar memoria.
ILS_E_FILTER_STRING	Cadena del filtro incorrecta.

- bstrFilterString: Cadena de texto que acaba en NULL describiendo la construcción del filtro.
- ppFilter: Dirección donde esté situado el puntero del objeto Filtro.

Descripción breve de lo que debe contener la cadena del filtro:

- Toma la forma ((Atributo=Valor) & (Atributo=Valor)).
- Los atributos se especifican como $#, donde # es el número ordinal en el tipo de enumerador ILS_STD_ATTR_NAME (por ejemplo, "$1", "$4", y así sucesivamente).
- Value es el valor que queremos que se corresponda con el atributo (Por ejemplo, "José", "López", etc.).
- El único valor para establecer concatenaciones es AND, especificado por &.
- Un ejemplo de cadena de filtro: (($1=jfk) & ($4=kennedy))

CREACIÓN DE UN OBJETO FILTRO

Existen dos formas de crear un objeto *Filtro* que son las siguientes:

- Construir el filtro de una pieza al mismo tiempo utilizando el método *CreateFilter* de la *API*.
- Construir una cadena representando el filtro y convertirlo a un objeto filtro utilizando el método *StringToFilter*.

CreateFilter tiene el prototipo siguiente:

```
HRESULT IIlsMain::CreateFilter(
[in] ILS_FILTER_TYPE FilterType;
[in] ILS_FILTER_OP FilterOp;
[out] IIlsFilter **ppFilter);
IIlsMain::CreateFilter
HRESULT IIlsMain::CreateFilter(
     ILS_FILTER_TYPE  FilterType,
     ILS_FILTER_OP  FilterOp,
     IIlsFilter **ppFilter);
```

StringToFilter tiene el siguiente prototipo:

```
HRESULT IIlsMain::StringToFilter(
                    [in] BSTR
    bstrFilterString,
                    [out] IIls-
    Filter **ppFilter);
```

Este método toma una expresión regular para la cadena de texto del filtro y lo convierte a un objeto *Filtro*. la expresión se parecería a:

```
$1=* & $5=San & ($10=ms-netmee-
    ting | $13=h323) & $4!=José &
    $6=Salamanca
```

La aplicación también puede agregar paréntesis sobre las relaciones para determinar una mínima claridad y comprensión de las operaciones del filtro, tal y como se muestra a continuación:

```
($1=*)&(($5=San) & (($10=ms-net-
    meeting) | ($13=h323)) &
    ($4!=José) & ($6=Salamanca))
```

Aquí **$1** indica el atributo del identificador de usuario, **$5** es el apellido, **$10** la aplicación, **$13** es el identificador de protocolo, **$4** es el nombre y **$6** es el nombre de la ciudad.

Se puede construir un filtro de una sola pieza utilizando el método CreateFilter

Los atributos estándar están indexados porque la aplicación no conoce los nombres de atributos del servidor. Los índices son definidos como *ILS_STD_ATTR_NAME*. Los valores de atributos estándar se definen en el archivo **Ils.idl** y se listan con el método *SetStandarAttribute*. Todas las cadenas de búsqueda deben incluir (**$1="patrón"**) para informar al servidor de que estamos interesados en recibir información de los usuarios. La mayoría utilizarán un patrón de (**$1=***) para devolver todos los usuarios cuyos atributos se correspondan con la cadena.

CREANDO FILTROS SIMPLES

Los siguientes métodos se utilizan para establecer los atributos de un filtro simple una vez que éste ha sido creado.

```
*HRESULT IIlsFilter::SetStan-
    dardAttributeName(
[in] ILS_STD_ATTR_NAME usrdAttr);
```

SetStandardAttributeName establece un atributo de nombre estándar. Los nombres serán índices del tipo *ILS_STD_ATTR_NAME*.

```
HRESULT IIlsFilter::SetExtendedAt-
    tributeName(
[in] BSTR bstrAnyAttrname);
```

SetExtendedAttributeName establece un atributo de nombre arbitrario. Debe ser un índice numérico.

```
HRESULT IIlsFilter::SetAttribute-
    Value(
[in] BSTR bstrAttrValue);
```

SetAttributeValues establece un valor objetivo. Todos los valores son *BSTR*. Este código demuestra cómo crear uno simple y utilizarlo para localizar usuarios con el apellido **"Sánchez"**.

```
// Crea un objeto de filtro sim-
    ple.
hr = CreateFilter(ILS_FITLERTY-
    PE_SIMPLE,
    ILS_FILTEROP_EQUAL, &pFil-
    ter);
if(SUCCEEDED(hr) {
    // Establece el nombre del
    filtro.
    hr = pFilter->SetStandardAt-
    tributeName(ILS_STDATTR_LAST_
    NAME);
    if(SUCCEEDED(hr) {
        // Establece el valor
    del filtro.
        LPTSTR_to_BSTR(&bstrLast-
    Name, "Sánchez");
        hr=pFilter->SetStan-
    dardAttributeValue( bstrLast-
    Name );
        ...
        // Utiliza el filtro que
    hemos creado para enumerar
    usuarios.
        hr = EnumUsers(pServer,
    pFilter, NULL, NULL, &uRe-
    qID);
        ...
    }
}
```

CREANDO FILTROS COMPUESTOS

Los siguientes métodos son utilizados para crear filtros compuestos.

```
HRESULT IIlsFilter::AddSubFilter
    (
[in] IIlsFilter *pFilter );
```

AddSubFilter agrega un filtro a la lista de subfiltros en uno compuesto. El orden de los subfiltros no es importante.

```
HRESULT IIlsFilter::RemoveSubFil-
    ter (
[in] IIlsFilter *pFilter );
```

RemoveSubFilter elimina el filtro especificado de la lista de subfiltros del filtro compuesto.

```
HRESULT IIlsFilter::GetCount (
[out] ULONG *pcElements );
```

GetCount cuenta el número de subfiltros contenidos en un filtro compuesto. El siguiente código crea un filtro compuesto que puede ser utilizado para buscar un usuario llamado **"Constantino Sánchez"**.

```
// Creamos el primer objeto
    filtro.
hr = CreateFilter(
ILS_FILTERTYPE_SIMPLE,
```

Tabla 5. Método AddSubFilter.

```
IIlsFilter::AddSubFilter
HRESULT IIlsFilter::AddSubFilter(
    IIlsFilter  *pFilter);
```

Agrega un subfiltro a la lista de subfiltros dentro de un filtro compuesto.

- Devuelve uno de los siguientes valores:

S_OK	Operación realizada.
ILS_E_FILTER_TYPE	El tipo de fichero no es ILS_FILTERTYPE_COMPOSITE.
ILS_E_POINTER	Subfiltro no válido.
ILS_E_MEMORY	No se pudo localizar memoria.

- pFilter: dirección del subfiltro que será abierto. Este método se aplica a operaciones de filtros compuestos.

```
ILS_FILTEROP_EQUAL,
&pFilterOne);
if(SUCCEEDED(hr)) {
// Establecemos el nombre del
    filtro.
hr = pFilterOne->SetStandardAt-
    tributeName(ILS_STDATTR_LAST_
    NAME);
if(SUCCEEDED(hr)) {
// Establecemos el valor del fil-
    tro.
LPTSTR_to_BSTR(&bstrLastName,
    "Sánchez");
hr=pFilterOne->SetStandardAttribu-
    teValue( bstrLastName );
...
    }}

// Crea el segundo objeto filtro
    simple.
hr = CreateFilter(ILS_FILTERTY-
    PE_SIMPLE,
ILS_FILTEROP_EQUAL,
&pFilterTwo);
if(SUCCEEDED(hr)) {
// Establecemos el nombre filtro.
hr = pFilterTwo->SetStandardAt-
    tributeName(
ILS_STDATTR_FIRST_NAME);
if(SUCCEEDED(hr)) {
// Establecemos el valor del
    filtro.
LPTSTR_to_BSTR(&bstrFirstName,
    "Constantino*");
hr=pFilterTwo->SetStandardAttribu-
    teValue(bstrFirstName );
...}
}
// Creamos el objeto de filtro
    compuesto.
hr = CreateFilter(ILS_FILTERTY-
    PE_COMPOSITE,
ILS_FILTEROP_AND, &pFilterComp);
if(SUCCEEDED(hr)){
// agregamos los subfiltros al
    filtro compuesto.
hr = pFilterComp-
    >AddSubFilter(pFilterOne);
 // Nota – Chequeamos sucesos
    aquí.
hr = pFilterComp-
    >AddSubFilter(pFilterTwo);
// Nota –Chequeamos sucesos
    aquí.}
// Usamos el filtro que hemos
    creado para enumerar los
    usuarios.
hr = EnumUsers(
        pServer,
        pFilterComp, NULL, NULL,
        &uReqID);
```

El código anterior es largo y a su vez complicado. Todo se puede simplificar utilizando el método *StringToFilter*, tal y como sigue:

```
// Creamos la cadena.
LPTSTR szFilter[] =
    "(%lu = *) & (%lu = %s) &
    (%lu = %s)";
wsprintf(
    szFilter,
    ILS_STDATTR_USER_ID,
    ILS_STDATTR_FIRST_NAME,
    "Constantino",
    ILS_STDATTR_LAST_NAME,
    "Sánchez");
// Convertimos la cadena.
LPTSTR_to_BSTR(&bstrFilter, szFil-
    ter);
hr = StringToFilter(bstrFilter,
    &pFilter)
// Usamos el filtro que hemos
    creado para enumerar los
    usuarios.hr = EnumUsers(
        pServer,
        pFilter,
        NULL, NULL,
        &uReqID);
```

La creación de filtros se puede complicar tanto como uno quiera. Construirlos no es difícil, aunque para filtros compuestos hay que poner atención en los bucles que contienen los simples. Si uno de estos últimos falla, el compuesto también lo hará.

REFERENCIA DE LA SERIE

Desarrollo de aplicaciones con videoconferencia

Sólo Programadores 55
INTRODUCCIÓN, ESTÁNDARES Y COMPONENTES

Sólo Programadores 56
PROYECTO DE CONFERENCIA MEDIANTE EL USO DE SCRIPTS DE VISUAL BASIC

Sólo Programadores 57
PROGRAMACION DE NETMEETING CON C++



XML (II). El lenguaje XSL

Adolfo Aladro García (aaladro@arrakis.es)

En el presente artículo analizaremos dos de las formas que tenemos para presentar los datos XML de una página HTML, las hojas de estilo en cascada (CSS, Cascade Style Sheet) o el lenguaje XSL (Extensible Stylesheet Language).

PRESENTACIÓN Y CONTENIDO

El estándar *XML* sirve para separar la presentación de los contenidos, con todas las ventajas que esto conlleva. Sin embargo, a pesar de esta dicotomía, suele ser necesario disponer de un mecanismo que sirva para visualizar la información *XML*. Es decir, un nexo de unión entre la presentación de los contenidos y los contenidos mismos. Las hojas de estilo en cascada y los *scripts* realizados en lenguaje *XSL* vienen a cubrir esta necesidad.

La presentación de los datos *XML* se hará a través de páginas *Web*, es decir, utilizando lenguaje *HTML*. Pero *HTML* es un lenguaje de etiquetas y por lo tanto se puede considerar que no es más que *XML*. Esto puede inducir a confusión, sobre todo porque *XSL* también es en realidad *XML*. Debemos tener claro qué representa cada uno, para qué sirven y cómo se combinan para dar lugar a una aplicación *Web*.

En general existen dos formas de presentar los datos procedentes de un documento *XML*: con hojas de estilo en cascada *(CSS)* o a través de hojas de estilo *XSL*. La primera de ellas es probablemente la más sencilla aunque no ofrece muchas posibilidades.

Para documentos *XML* en el que los datos tengan una estructura sencilla y regular, puede que las hojas de estilo *CSS* sean más que suficientes. El lenguaje *XSL* es algo más complejo pero pone a nuestra disposición toda la flexibilidad y potencia de un verdadero mecanismo de consulta de datos.

LAS HOJAS DE ESTILO EN CASCADA

Imaginemos que en un documento *HTML* aparecen una serie de titulares, éstos han de aparecer así:

```
<FONT FACE="Courier New"
    COLOR="Red">
<B>Esto es un titular...</B>
</FONT>
```

El anterior código habrá de repetirse para cada titular que aparezca en el documento *Web* y su significado es evidente: los titulares se presentarán con letra del tipo **Courier New**, de color negro y en negrita. Si un día se decide cambiar el aspecto del sitio *Web* y de repente los titulares han de aparecer con otros atributos de presentación, será necesario ir uno a uno retocando el correspondiente código *HTML*.

HTML es un lenguaje de etiquetas, se puede considerar que es XML

¿No sería mucho mejor tener un sistema que permitiera guardar los estilos de los elementos? Si fuera así, cada vez que tuviéramos que cambiar la presentación de una serie de elementos, solamente tendríamos que cambiar el estilo de ese tipo de elementos. Esto en definitiva es lo que suponen las hojas de estilo. Una hoja de estilo es una definición de atributos de código *HTML*. Siguiendo con el ejemplo anterior, podríamos definir la siguiente hoja de estilo:

```
.titular {
    font-family : Courier New;
  font-weight : bold;
```

```
  color : Red;
}
```

Ésta la guardaríamos tal cual en un archivo con extensión *.css* (por ejemplo, **miestilo.css**) y añadiríamos en la página *HTML* el código siguiente en la cabecera (entre *<HEAD>* y *</HEAD>*):

```
<LINK REL="STYLESHEET"
     TYPE="text/css"
     HREF="miestilo.css">
```

A partir de este momento los titulares aparecerían de la siguiente forma:

```
<DIV CLASS="titular">
Esto es un titular...
</DIV>
```

Bastaría con cambiar la hoja de estilo para que todos los titulares modifiquen automáticamente su aspecto sin retocar ninguna página *HTML*.

No es el propósito de este artículo adentrarnos en profundidad en todos los aspectos relacionados con las hojas de estilo en cascada (véase **http://msdn.microsoft.com/workshop/author/css/reference/attributes.asp** para conocer todos los atributos disponibles y sus posibles valores). Partiendo del ejemplo anterior simplemente tenemos que observar que existen una serie de atributos definidos, que pueden tomar distintos valores, y que sirven para determinar la forma en la que se visualizarán los elementos. Cuando reunimos un conjunto de atributos creamos un estilo, y éste puede ser aplicado a cualquier elemento.

El listado 1 ofrece el código fuente del documento *XML* **discos.xml**. En él se presenta una pequeña base de datos relativa a una colección de discos. El listado 2 muestra la hoja de estilo *CSS* que utilizamos para mostrar los datos y el resultado puede apreciarse en la figura 3.

Las etiquetas *XML* no asumen ningún formato de salida, fuentes o colores. Toda esta información ha de ser aportada por la hoja de estilo y ésta es referenciada en el documento *XML* con la siguiente instrucción:

```
<?xml-stylesheet type="text/css"
href="discos.css" ?>
```

Cuando el navegador intenta cargar el documento *XML* busca esta instrucción, descarga la hoja de estilo correspondiente y la utiliza para visualizar el documento. Nótese que la instrucción se usa automáticamente sólo cuando se intenta visualizar un documento *XML* directamente en el navegador. Si los datos *XML* están contenidos en una página *HTML* se ignora la instrucción relativa a la hoja de estilo. Toda instrucción de hoja de estilo debe tener el atributo *type*. Los posibles valores que puede tomar este atributo determinan el tipo de hoja de estilo que ha de aplicarse: **"text/css"** indica una hoja de estilo *CSS* y **"text/xsl"** indica una hoja de estilo *XSL*. El atri-

Figura 1. En la imagen se muestra el documento discos.xml visualizado por el navegador sin tener ninguna hoja de estilo asociada.

Listado 1. Documento XML contenido en el archivo discos.xml.

```
<?xml version='1.0'?>
<discos>
   <disco id="45878952">
      <artista>Madonna</artista>
      <titulo>Ray Of Light</titulo>
      <anyo>1994</anyo>
   </disco>
   <disco id="36567552">
      <artista>Madonna</artista>
      <titulo>Erotica</titulo>
      <anyo>1992</anyo>
   </disco>
</discos>
```

buto *href* es una dirección *URL* que apunta a la hoja de estilo.

Un documento *XML* puede tener múltiples instrucciones de procesamiento. En tal caso, el navegador busca la primera instrucción de procesamiento de hoja de estilo del tipo **"text/xsl"** y la utiliza. Si no la encuentra entonces busca las hojas de estilo del tipo **"text/css"** y las aplica en cascada.

Las etiquetas XML no asumen ningún formato de salida, fuentes o colores

El archivo *CSS* correspondiente contendrá tantos selectores como deseemos y el nombre de éstos coincidirá con el de la etiqueta *XML* para la que queremos definir una serie de atributos de visualización. Uno de los más importantes es el que determina la propiedad *display*. La tabla 1 muestra un resumen de los distintos valores que ésta puede tomar. Para los datos *XML*, el valor por defecto de la propiedad *display* es **inline**.

Listado 2. Hojas de estilo CSS que sirve para presentar el documento XML discos.xml.

```
discos  {
  display : block;
}
disco  {
  display : block;
  padding : 10px;
  background-color : Silver;
  margin : 10px;
}
artista  {
  display : block;
}
titulo  {
  color : Olive;
  font-weight : bold;
}
anyo  {
  color : Maroon;
  font-weight : bold;
}
```

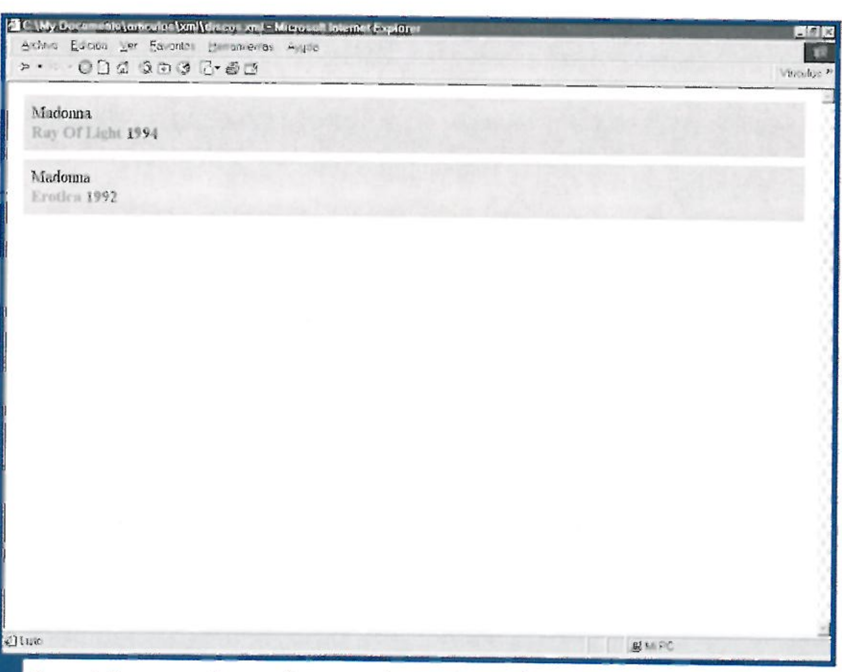


Figura 2. Documento discos.xml visto por el navegador con la hoja de estilo discos.css. asociada.

EL LENGUAJE XSL

El lenguaje *XSL* (*Extensible Stylesheet Language*) nace para dar respuestas a las siguientes necesidades que plantea la tecnología *XML*:

- **Presentación de los datos XML.** El lenguaje *XSL* permite visualizar los datos *XML* ya que transforma estos datos en código *HTML* listo para ser visualizado por cualquier navegador. Al tener control absoluto en ese proceso de transformación, se puede personalizar la presentación de la información haciéndola tan sofisticada como se desee.
- **Navegación directa de los archivos XML.** Los navegadores podrán utilizar hojas *XSL* para hacer posible la navegación directa de archivos *XML*. Estas hojas le dirán al navegador cómo han de presentarse los datos.
- **Transformación de la información (consultas, búsquedas y filtrado de datos).** *XSL* realiza con-

sultas, búsquedas y filtrado de datos a partir de un documento *XML* dado.

Las hojas de estilo en cascada no son suficientes a la hora de visualizar los datos *XML*. Para que así sea la estructura de estos datos debería ser siempre virtualmente idéntica a la estructura de la presentación. Esto es lo que ocurría en el ejemplo anterior que hemos visto.

El lenguaje XSL permite visualizar los datos XML

Conceptualmente no existe ninguna diferencia entre la estructura del archivo **discos.xml** y la disposición de los distintos elementos cuando se visualizan utilizando la hoja de estilo **discos.css**. ¿Qué ocurriría si quisiéramos, por ejemplo, ordenar los datos antes de presentarlos? ¿o si por ejemplo solamente deseáramos mostrar los discos cuyo año de publicación se encuentra comprendido entre 1996 y 1999? Evidentemente ninguna de estas cosas pueden realizarse con hojas de estilo *CSS*. Existirán muchos documentos *XML* y será muy difícil poder visualizarlos con *CSS*. El lenguaje *XSL* va mucho más allá permitiendo elaboradas transformaciones a partir de los datos *XML*.

A diferencia de lo que ocurre con las hojas de estilo *CSS*, el lenguaje *XSL* transforma la estructura de datos *XML* en otra nueva lo que permite reordenar los elementos, generar texto adicional y realizar cálculos, todo ello sin tener porqué modificar la fuente de datos *XML*. Las hojas *XSL* contienen una plantilla con la estructura resultado que se desea y es posible identificar los datos de la fuente original de forma que podamos insertarlos en la plantilla. Aquellos que conozcan el fundamento de las páginas *ASP* observarán rápidamente que el modelo que plantea el lenguaje *XSL* es muy similar.

Listado 3. Documento XML contenido en el archivo contactos.xml.

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="contactos.xsl" ?>
<contactos>
  <contacto id="123456">
    <nombre>El Peque&#241;o Tim</nombre>
    <telefono>79624538</telefono>
    <email>elpeque&#241;otim@spp.es</email>
  </contacto>
  <contacto id="7890123">
    <nombre>Casa de Mu&#241;ecas</nombre>
    <telefono>25465465</telefono>
    <email>casademu&#241;ecas@spp.es</email>
  </contacto>
  <contacto id="456789">
    <nombre>Por Amor al Arte</nombre>
    <telefono>65465456</telefono>
    <email>poramoralarte@spp.es</email>
  </contacto>
</contactos>
```

XSL es capaz de manejar todo tipo de datos. Es posible definir fragmentos de plantillas, y el procesador las combina todas de forma tal que el resultado final refleja la estructura del documento *XML* original. Cada fragmento de plantilla declara el tipo y el contexto de los nodos fuentes para los que son apropiados, por lo que el procesador *XSL* podrá asociar nodos con fragmentos de plantillas. Supongamos que partimos del documento *XML* contenido en el archivo **contactos.xml** (ver listado 3) y queremos realizar un hoja de estilo *XSL*. El listado 4 muestra el código fuente de la hoja de estilo *XSL* correspondiente. Se trata de una combinación de código *HTML* con otro tipo de etiquetas para acceder a los datos del documento *XML*. Veamos su significado:

EL ELEMENTO XSL:FOR-EACH

Permite la aplicación del mismo patrón a varios nodos. La sintaxis de este elemento es la siguiente:

```
<xsl:for-each order-by="criterio-
    ordenación" select="patrón" >
```

- El atributo *order-by* es una cadena de caracteres que contiene una lista de criterios de ordenación separados por el símbolo ";". Los criterios se van aplicando en el orden en el que aparecen. Al tratar de ordenar dos elementos si el primer criterio no proporciona ninguna información al respecto, automáticamente se intenta aplicar el segundo criterio y así sucesivamente. El primer carácter distinto de blanco en cada criterio determina si se sigue un orden ascedente "+" o descendente "-". El criterio de ordenación se expresa como un patrón *XSL* relativo al patrón descrito en el atributo *select*.
- El atributo *select* es el patrón *XSL* que se utilizará como criterio de selección dentro del contexto actual. El valor por defecto "**node**()" indica la evaluación del nodo actual.

EL ELEMENTO XSL:VALUE-OF

Evalúa un patrón *XSL* determinado por el atributo *select* y devuelve el valor del elemento solicitado

como texto, que se inserta dentro de la plantilla.

```
<xsl:value-of select="patrón"/>
```

El valor por defecto del atributo *select* es ".", lo que equivale a decir "el valor del nodo actual".

EL ELEMENTO XSL:TEMPLATE

Define una plantilla que se va a utilizar para generar la presentación de un determinado nodo fuente.

```
<xsl:template language="lenguaje"
    match="contexto" >
```

- El atributo *language* define el lenguaje de *scripting* que puede utilizarse en esta hoja de estilo para extender las capacidades de *XSL*.

Las hojas de estilo en cascada no son suficientes para visualizar los datos XML

- El atributo *match* define el contexto para el que se ha creado la plantilla. Puede usarse para cambiar el contexto del documento fuente y proporcionar una manera de navegar por el árbol del documento. El valor por defecto es *node()* | / | @*; para todos los nodos.

EL ELEMENTO XSL:STYLESHEET

Define el conjunto de plantillas aplicables al árbol de datos fuente para generar otro árbol de salida.

```
<xsl:stylesheet default-
    space="preserve" indent-
    result="yes" language="len-
    guaje" result-ns="valor" >
```

- El atributo *default-space* determina si se preservan los espacios en blanco del documento fuente. Sólo soporta el valor **"default"** (los demás se ignoran).
- El atributo *indent-result* indica si se han de preservar en la salida cualquier carácter en blanco que aparezca en la hoja de estilo. Solamente se admite el valor **"yes"** (todos los demás valores se ignoran).
- El atributo *language* determina el lenguaje de *scripting* que se utiliza en la hoja de estilo.
- El atributo *result-ns* indica qué tipo de salida produce el procesador *XSL*. En *Explorer 5* todas las salidas son *XML*, incluyendo documentos *HTML* bien formados, por lo que este atributo se ignora.

Listado 4. Hojas de estilo XSL que sirve para presentar el documento XML contactos.xml.

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
<BODY>
<TABLE BORDER="1" CELLSPACING="4" CELLPADDING="4">
<TR>
<TD>Nombre</TD>
<TD>Telefono</TD>
<TD>Nombre</TD>
</TR>
<xsl:for-each select="contactos/contacto">
<TR>
<TD>
<xsl:if test=".[@id='456789']">
<xsl:attribute name="BGCOLOR">#CCCCCC</xsl:attribute>
</xsl:if>
<xsl:value-of select="nombre"/>
</TD>
<TD><xsl:value-of select="telefono"/></TD>
<TD><xsl:value-of select="email"/></TD>
</TR>
</xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

DOCUMENTOS HTML BIEN FORMADOS

Si observamos el código fuente contenido en **contactos.xsl** es fácil darse cuenta de que una hoja de estilo *XSL* representa en sí misma un documento *XML*: tiene sus etiquetas (las propias y las del lenguaje *HTML*), éstas se disponen siguiendo las normas que vimos para el estándar *XML*, etc. De ahí que si cargamos directamente el archivo **contactos.xsl** en el navegador obtengamos un resultado similar al que logramos cada vez que cargamos un archivo *XML* para el cual no hemos definido ninguna hoja de estilo de presentación.

El hecho de que las hojas de estilo *XSL* sean en sí mismas documentos *XML* conlleva innumerables ventajas. La más evidente es que si el navegador ha sido reconstruido para soportar *XML*, el soporte para *XSL* se puede considerar que ya está medio hecho también.

Una hoja de estilo XSL representa en sí misma un documento XML

Normalmente en las hojas de estilo *XSL* se intercalarán trozos de código *HTML*. Para garantizar que las hojas de estilo *XSL* son documentos *XML* correctos es necesario tener constancia de que el código *HTML* esté bien formado, de acuerdo a las normas del estándar *XML*. Por lo tanto, un documento *HTML* bien formado es aquel que cumple las reglas que impone el estándar *XML*. Esto supone que hemos de utilizar las etiquetas *HTML* conocidas pero con la sintaxis de *XML*. En el primero de los artículos de esta serie vimos las reglas más importantes que debe cumplir todo documento *XML* para poder ser considerado como bien formado. Vamos a recordarlas brevemente:

- **Todas las etiquetas deben cerrarse.** Esto quiere decir que si utilizamos la etiqueta *<P>*, por ejemplo, debemos siempre utilizar la etiqueta de cierre *</P>* (en un página *HTML* "normal" esto no es necesario). Además, las etiquetas que no la tienen de cierre deben llevar el carácter "/" antes del carácter ">" que marca el final. Por ejemplo: *
* o *<IMG SRC="madonna.gif" />*.
- **El orden de apertura y cierre de las etiquetas debe ser el correcto.** El anidamiento de las etiquetas debe ser correcto. Por ejemplo, no sería válido hacer: *<B><I>madonna</B></I>*. Lo que habría que escribir sería: *<B><I>madonna</I></B>* o *<I><B>madonna</B></I>*.

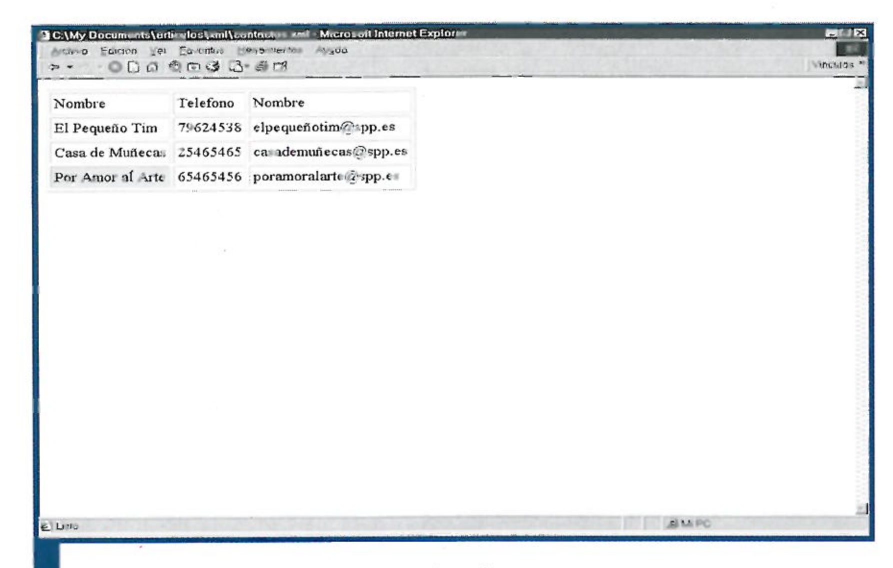


Figura 3. Visualización directa de una hoja de estilo.

- **Se diferencia entre mayúsculas y minúsculas.** No es lo mismo escribir la etiqueta *<TABLE>* que *<table>*, o que *<tABLe>* .
- **Los atributos deben ir siempre entre comillas.** Nunca debe escribirse algo como *<TABLE WIDTH=100 BORDER=0>*, sino *<TABLE WIDTH="100" BORDER="0">* (de nuevo hacemos notar que la primera de las opciones es totalmente válida en una página *HTML* normal y corriente, no así en una hoja de estilo *XSL*).
- **Solamente puede haber una etiqueta raíz.** Las etiquetas *<HTML>* y *</HTML>* tienen que contener todo el documento.
- **Entidades de caracteres.** La entidades de caracteres sirven para referirnos a los caracteres. Así por ejemplo, el carácter "<" se escribe dentro de un documento como *<*. Como ya vimos, el estándar *XML* solamente define unas pocas entidades de caracteres pero soporta también la entidades numéricas de forma que, por ejemplo, el carácter "ñ" se debe escribir dentro de un documento como *ñ* (ver **http://msdn.microsoft.com/workshop/author/dhtml/reference/charsets/charsets.asp**)
- **Bloques de código.** Los bloques de código *script* (típicamente *JavaScript*) dentro de una página *HTML* pueden contener caracteres que impidan un correcto proceso de *parsing* del documento, por lo que deberán ser escapados para realizar un documento *HTML* bien formado, o bien el código de *script* se incluirá dentro de una sección *CDATA*. Por ejemplo, el siguiente bloque de código:

```
<SCRIPT>
alert("Esto es código JavaScript");
</SCRIPT>
```

debería escribirse como:

```
<SCRIPT><![CDATA[
  alert("Esto es JavaScript");
]]></SCRIPT>
```

CÓMO ACCEDER A LOS ATRIBUTOS

Los atributos que aparecen en las etiquetas de un documento *XML* fuente pueden ser accedidos dentro de patrones *XSL* situando el símbolo "@" delante del nombre del atributo.

```
<xsl:value-of select="@atributo"/>
```

Además *XSL* es capaz de generar atributos que se incorporan a *HTML*.

```
<xsl:attribute name="ATRIBUTO">
<xsl:value-of select="patrón"/>
...
</xsl:attribute>
```

Se pueden crear atributos de dos formas: colocándolos dentro de un elemento de salida, como por ejemplo *<TABLE BORDER="0">*; o añadiéndolos mediante el elemento *<xsl:attribute>*, que permite generar un atributo a partir de los datos *XML* de entrada. Pero debemos tener en cuenta las siguientes limitaciones:

- No se puede añadir un atributo a un elemento que ya tiene un atributo con ese mismo nombre.
- Los atributos que se añaden con el elemento *<xsl:attribute>* deben aparecer antes del hijo.

El la hoja de estilo **contactos.xsl** podemos encontrar un ejemplo. En ella se genera dinámicamente el atributo *BGCOLOR* de una etiqueta *<TD>* de una tabla. Es más, esto se hace dependiendo del valor de que tenga el atributo *id* del contacto para cada caso.

PLANTILLAS CONDICIONALES

Las plantillas condicionales sirven para generar código *HTML* dependiendo de que se cumplan unas ciertas condiciones en el documento fuente. *XSL* proporciona los elementos *<xsl:if>* y *<xsl:choose>* para poder desarrollar plantillas condicionales.

EL ELEMENTO XSL:IF

Permite fragmentos de plantilla condicionales.

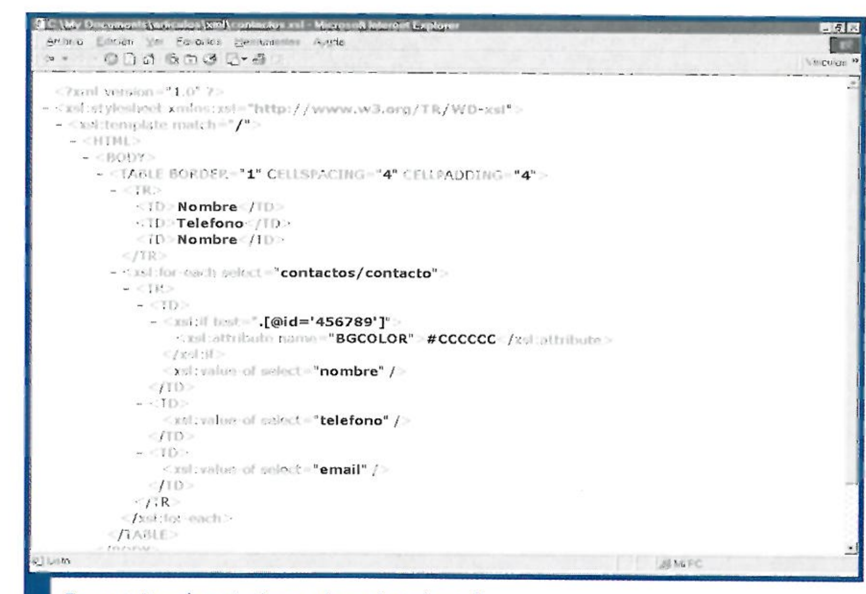


Figura 4. Visualización directa de una hoja de estilo.

```
<xsl:if expr="expresión" langua-
    ge="lenguaje"  test="patrón-
    condicional" >
```

- El atributo *expr* es una expresión cuya evaluación da como resultado un valor booleano. Si la expresión devuelve **true** y el patrón que alberga *test* es también **true**, el patrón tiene éxito y los contenidos asociados al elemento aparecerán en la salida.
- El atributo *language* hace referencia al lenguaje utilizado en la expresión *expr*.
- El atributo *test* representa la condición que ha de verificarse en los datos. Esta condición vienen dada en forma de patrón *XSL*. En el momento en el que haya un nodo del árbol de datos que cumpla el patrón, los contenidos asociados aparecerán en la salida.

LOS ELEMENTOS XSL:CHOOSE, XSL:OTHERWISE Y XSL:WHEN

El elemento *xsl:choose* proporciona una forma de verificar múltiples condiciones. Ha de usarse en conjunción con los elementos *xsl:otherwise* y *xsl:when*.

```
<xsl:choose>
<xsl:when test=".[@id='456789']">
<xsl:attribute
    name="BGCOLOR">#CCCCCC</xsl:a
    ttribute>
</xsl:when>
<xsl:otherwise>
```

Tabla 1. Resumen de los valores más importantes que puede tomar la propiedad display.

Propiedad	Significado
block	El objeto se muestra como un bloque diferenciado (con un salto de línea antes y después).
none	El objeto no se visualiza.
inline	El objeto se muestra siguiendo el flujo normal de los elementos que le rodean sin que se produzcan saltos de línea.
list-item	El objeto se muestra como un bloque y además se añade un marcador a modo de lista.

```
<xsl:attribute
     name="BGCOLOR">#FFCCCC</
     xsl:attribute>
</xsl:otherwise>
</xsl:choose>
```

PATRONES XSL

Los patrones *XSL* representan una manera de tener acceso rápido a los nodos de un documento *XML*. Pueden identificar los nodos basándose en su tipo, su nombre, sus valores, y la relación que existe entre ese nodo y otros dentro del mismo documento. Un patrón *XSL* puede describirse como un camino dentro del árbol *XML*. Éste está formado por una lista de elementos separados por el carácter "/". Por ejemplo: **/contactos/contacto/nombre**

El patrón *XSL* se verificará para todos los elementos que se encuentren en el camino descrito. Además el lenguaje *XSL* dispone de caracteres especiales que describen ciertos tipos de elementos. Por ejemplo, un elemento que puede tener cualquier nombre se representa como "*": **/contactos/* /nombre**. El patrón anterior identifica igualmente a todos los nombres de la lista de contactos pero esta vez no tienen porqué ser los elementos 'nombre' hijos de un elemento 'contacto'.

También se pueden especificar bifurcaciones utilizando corchetes. El siguiente patrón identificaría a todos los nombres siempre y cuando el elemento 'contacto' correspondiente tuviera un elemento hijo llamado **'email'**: **/contactos/contacto [em ail]/nombre**

Además podríamos añadir comparaciones. Nótese que las comparaciones siempre han de ir encerradas entre corchetes: **/contactos/contacto[em ail= 'poramoralarte@spp.es']/nombre.**

Tabla 2. Resumen de operadores y caracteres especiales que se emplean en los patrones XSL

Carácter especial u operador	Significado
/	Operador "nodo hijo". Selecciona el nodo hijo inmediato de la colección de la izquierda. Cuando este operador aparece al principio del patrón indica que los hijos deben ser seleccionados a partir del nodo raíz.
//	Operador "descenso recursivo". Busca el elemento especificado a cualquier nivel de profundidad dentro del árbol. Cuando este operador aparece al principio del patrón, indica que se va a realizar una búsqueda recursiva a partir del nodo raíz.
.	Indica el contexto actual.
*	Selecciona todos los elementos sin tener en cuenta su nombre.
@	Selecciona todos los atributos sin tener en cuenta su nombre.
[]	Aplica un patrón de filtro.

Los atributos se indican en los patrones mediante el carácter "@". Se pueden utilizar para realizar bifurcaciones (Todos los nombres de contactos en el que el atributo 'id' tiene un valor igual a '7890123' se efectuaría de la forma siguiente: **/contactos/contacto [@id="7890123"]/nombre**).

O bien simplemente para identificar nodos (Los nodos correspondientes al atributo 'id' se obtendrían: **/contactos/contacto/@id**).

La tabla 2 contiene un resumen de algunos de los principales operadores y caracteres especiales que pueden utilizarse en los patrones *XSL*. Por ejemplo, el carácter "." sirve para referirse al contexto actual. Por eso cuando en nuestro ejemplo, **contactos.xsl**, hacíamos la siguiente referencia dentro de un elemento *xsl:for-each*.

```
<xsl:if test=".[@id='456789']">
```

El patrón que en realidad se estaba verificando por cada vuelta del bucle era el siguiente.

```
/contactos/contacto[@id='456789']
```

En el presente artículo hemos introducido el lenguaje *XSL*, un estándar realmente potente que viene a complementar a la tecnología *XML*. En los sucesivos capítulos profundizaremos en muchos de los aspectos que por el momento sólo hemos podido tratar de manera superficial, como por ejemplo el mecanismo de patrones *XSL*, y además veremos como es posible integrar las hojas de estilo *XSL* dentro de las páginas *HTML* que contienen islas de datos. Hasta el mes que viene.

REFERENCIA DE LA SERIE

XML

Sólo Programadores 56
INTRODUCCIÓN A XML

Sólo Programadores 56
CÓMO TRABAJAR CON EL DOM

Programación del registro de Windows (I)

Adolfo Aladro (aaladro@arrakis.es)

El objetivo de esta serie consiste en aprender la forma de acceder al registro de *Windows* desde nuestros propios programas, algo indispensable para cualquier desarrollador que desee realizar aplicaciones profesionales.

El registro de *Windows* es una base de datos donde se guarda de forma permanente todo tipo de información. Cuando *Windows* se inicializa, o una aplicación cualquiera se ejecuta, normalmente se accede al registro en busca de una serie de parámetros. Las aplicaciones de *Windows* poco a poco han ido abandonando casi por completo el antiguo mecanismo de archivos *INI* de inicialización para pasar a almacenar la información de su configuración en el citado registro.

Como ya sabemos, esta *API* puede invocarse desde un programa hecho en *C*, *C++*, *Visual C++*, *Visual Basic* y un largo etcétera.

En nuestro caso concreto hemos optado por utilizar *Visual Basic* ya que ofrece un entorno de programación sencillo y sin demasiadas complicaciones. En cualquier caso todos los conocimientos que aquí expongamos podrán ser aplicados a cualquier otro lenguaje de programación ya que se trata de la *API* de *Windows*, que evidentemente es siempre la misma con independencia del lenguaje de programación elegido.

El registro es una base de datos en la que se guarda todo tipo de información permanentemente

Tan importante es saber programar procedimientos que accedan al registro como conocerlo, su estructura, contenido, y especialmente, cuáles son aquellas características de *Windows* que se ocultan allí y cómo modificarlas. El registro contiene prácticamente toda la información relativa a la configuración de nuestro sistema. Algunos de estos elementos pueden ser modificados desde aplicaciones que el propio *Windows* pone a nuestra disposición, pero algunos otros no se han hecho visibles al usuario. A lo largo de esta serie de artículos también haremos ejemplos que muestren algunas de estas características "ocultas" de *Windows*.

ORGANIZACIÓN DEL REGISTRO

El registro de *Windows* está organizado jerárquicamente en forma de árbol. La raíz del árbol es nuestro ordenador y partiendo de esta raíz existen una serie de ramas cada una de las cuales está destinada a guardar cierto tipo de información. A cada rama se la denomina clave o entrada del registro y puede contener a su vez otras claves o valores. Existen una serie de claves predefinidas que se encuentran en lo alto de la jerarquía.

- *HKEY_CLASSES_ROOT*. Almacena información de las clases *OLE* del sistema, asociación de archivos y otra información relativa a la interfaz de *Windows*.

- *HKEY_CURRENT_USER.* Guarda la información relativa a todos los aspectos que afectan a las preferencias del usuario: variables de entorno, datos de los grupos de programas, preferencias de aplicaciones, etc. Los programas que instalamos en nuestro ordenador normalmente utilizan esta clave para guardar datos específicos.
- *HKEY_LOCAL_MACHINE.* Aquí se almacenan datos que tienen que ver con las particularidades físicas del ordenador: memoria del sistema, dispositivos multimedia, tarjetas *hardware* instaladas, etc. También se encuentran los parámetros de configuración de los dispositivos, sus preferencias y todas las posibles opciones.
- *HKEY_USERS.* Esta clave contiene información específica para cada usuario del ordenador (donde cobra especial relevancia esta clave es en sistemas *Windows NT*).
- *HKEY_CURRENT_CONFIG.* Almacena junto con *HKEY_LOCAL_MACHINE* la información sobre los elementos físicos del ordenador así como otros datos de la configuración actual del sistema.
- *HKEY_DYN_DATA.* Finalmente esta clave, un tanto especial, guarda información acerca de los registros dinámicos de la máquina.

El programa *REGEDIT*, que se suministra con *Windows*, sirve para visualizar y modificar el registro (para acceder a él tenemos que ir al menú **Inicio** y seleccionar **Ejecutar**. Entonces teclearemos **regedit** y pulsaremos la tecla **Enter.**). La manipulación directa del registro por parte del usuario no es algo que se haga con frecuencia y de hecho no está aconsejada. El registro de *Windows* contiene información crítica para el sistema por lo que un error en el mismo puede tener consecuencias catastróficas para el ordenador. Esto no sólo nos afecta como usuarios, ya que como programadores debemos asegurarnos con toda garantía de que nuestros programas acceden al registro de forma correcta. Al mismo tiempo, durante el proceso de desarrollo de las aplicaciones tendremos que hacer múltiples pruebas en nuestro sistema a fin de comprobar que las aplicaciones creadas por nosotros mismos funcionan correctamente.

El registro de Windows está organizado jerárquicamente en forma de árbol

Por lo tanto, lo primero que es casi obligatorio hacer antes de nada es aprender a hacer una copia de seguridad del registro, de manera que si nuestro programa realiza alguna operación errónea podamos retornar a un estado anterior. Esta situación puede darse con relativa facilidad si es la primera vez que realizamos una aplicación de estas características o incluso aunque ya lo hayamos hecho más veces. Con frecuencia ningún programa funciona a la primera, y en este caso concreto un mal funcionamiento puede tener consecuencias nefastas por lo que es mejor asegurarse si no queremos vernos envueltos en un problema que nos obligue a instalar todo nuestro equipo desde el principio.

CÓMO REALIZAR UNA COPIA DE SEGURIDAD

El registro de *Windows* se encuentra físicamente en dos archivos dentro del propio directorio de *Windows*: **SYTEM.DAT** y **USER.DAT**. Si comprobamos ese mismo directorio veremos que también se encuentran los archivos **SYSTEM.DA0** y **USER.DA0** que contienen el registro correspondiente al estado en que se encontraba la última vez que se inició el ordenador sin problemas. Además de lo anterior, la aplicación *REGEDIT* permite exportar o importar el registro entero, o sólo algunas ramas. Los archivos que contienen información exportada del registro son archivos de texto con extensión *.REG*, y como tales archivos de texto pueden ser editados con cualquier procesador de textos. Aunque existen muchas herramientas capaces de realizar copias de seguridad del registro, podemos llevarlo a cabo nosotros mismos de forma sencilla con todas las garantías de seguridad para nuestro sistema. Existen dos métodos:

MÉTODO 1

1. Reiniciar el ordenador, presionar **F8** en el momento en el que aparezca el mensaje **Inciando Windows 95/98** y elegir la opción **Sólo símbolo de sistema del modo A prueba de fallos** en el menú de **Inicio** que aparece.
2. En el símbolo del sistema teclearemos lo siguiente:

```
CD WINDOWS
ATTRIB -S -H -R SYSTEM.DAT
ATTRIB -S -H -R USER.DAT
COPY SYSTEM.DAT SYSTEM.BAK
COPY USER.DAT USER.BAK
```

NOTA: S*e supone que Windows está instalado en el directorio "WINDOWS". Si en el directorio de Windows ya hubiera alguno de esos ficheros con extensión BAK elegiríamos cualquier otra extensión.*

3. Reiniciar el equipo.

Para restaurar estos ficheros del Registro en el caso de que los originales se hubieran perdido o dañado, se debe realizar lo siguiente:

1. Reiniciar el ordenador, presionar **F8** en el momento en el que aparezca el mensaje **Iniciando Windows 95/98** y elegir la opción **Sólo símbolo de sis-**

tema del modo A prueba de fallos en el menú de **Inicio** que aparece.

2. En el símbolo de sistema tecleamos lo siguiente:

```
CD WINDOWS
ATTRIB -S -H -R SYSTEM.DAT
ATTRIB -S -H -R SYSTEM.DA0
ATTRIB -S -H -R USER.DAT
ATTRIB -S -H -R USER.DA0
REN SYSTEM.DAT SYSTEM.XXX
REN SYSTEM.DA0 SYSTEM.XX1
REN USER.DAT USER.XXX
REN USER.DA0 USER.XX1
COPY SYSTEM.BAK SYSTEM.DAT
COPY USER.BAK USER.DAT
```

3. Reiniciar el equipo

MÉTODO 2

En el *CD-ROM* de *Windows 95/98*, dentro del directorio **\OTHER\MISC\ERU**, existe una utilidad llamada *ERU* (*Emergency Recovery Utility*) que sirve para crear una copia de seguridad de la configuración del sistema y los ficheros del registro.

Posteriormente podremos restaurar estos valores en caso de pérdida o daño. Cuando ejecutamos el programa debemos seguir los siguiente pasos:

1. Podemos guardar la copia de seguridad de configuración del sistema en disquetes o en un directorio del disco duro. Si elegimos la primera de las opciones necesitaremos más de un disquete. En el segundo caso es conveniente que seleccionemos un directorio reservado a tal efecto de forma que sepamos que ahí se encuentra la información.

En primer lugar es imprescindible, y casi obligatorio, aprender a realizar una copia de seguridad del registro

2. Se muestra una ventana donde aparecen los nombres de los ficheros del sistema que van a guardarse. Por defecto la herramienta hace copia de seguridad de los siguientes: **CONFIG.SYS, AUTOEXEC.BAT, WIN.INI, SYSTEM.INI, PROTOCOL.INI, USER. DAT, SYSTEM.DAT, IO.SYS, COMMAND.COM** y **MSDOS.SYS**. También existe la posibilidad de personalizar la copia de seguridad seleccionando sólo algunos de los ficheros anteriores.
3. Cuando presionamos el botón **Next** se procede a hacer la copia de seguridad en la ubicación seleccionada en el paso 1.
4. Finalmente se muestra una pantalla donde se nos comunica que la copia se ha realizado.

Para restaurar estos ficheros se deben seguir los siguientes pasos:

1. Reiniciar el ordenador, presionar **F8** en el momento en el que aparezca el mensaje **Iniciando Windows 95/98** y elegir la opción **Sólo símbolo de sistema del modo A prueba de fallos** en el menú de **Inicio** que aparece.
2. Ir al directorio donde elegimos almacenar la copia de seguridad o bien introducir el disquete en el caso de que seleccionáramos esta opción.
3. Ejecutar el programa *ERD.EXE*.

Listado 1. Procedimiento que se encarga de intentar abrir una entrada del registro de Windows.

```
Private Sub Command1_Click()
Dim lResultado As Long
Dim lIdClave As Long
Dim lClaveRaiz As Long
Dim sClave As String
If Combo1.ListIndex <> -1 Then
    Select Case Combo1.List(Combo1.ListIndex)
    Case "HKEY_CLASSES_ROOT"
        lClaveRaiz = HKEY_CLASSES_ROOT
    Case "HKEY_CURRENT_USER"
        lClaveRaiz = HKEY_CURRENT_USER
    Case "HKEY_LOCAL_MACHINE"
        lClaveRaiz = HKEY_LOCAL_MACHINE
    Case "HKEY_USERS"
        lClaveRaiz = HKEY_USERS
    Case "HKEY_CURRENT_CONFIG"
        lClaveRaiz = HKEY_CURRENT_CONFIG
    Case "HKEY_DYN_DATA"
        lClaveRaiz = HKEY_DYN_DATA
    End Select
    sClave = Text1.Text
    lResultado = RegOpenKeyEx(lClaveRaiz, sClave, 0,
                              KEY_ALL_ACCESS, lIdClave)
    If lResultado = ERROR_SUCCESS Then
        Label3.Caption = "La clave se abrió con éxito"
        lResultado = RegCloseKey(lIdClave)
    Else
        Label3.Caption = "No se pudo abrir la clave"
    End If
End If
End Sub
```

LA API DE WINDOWS

Proporciona una serie de procedimientos gracias a los cuales podemos manipular el registro desde nuestros programas. A lo largo de esta serie de artículos los iremos viendo prácticamente todos. En nuestro caso concreto, como vamos a utilizar *Visual Basic* para desarrollar los ejemplos, lo primero que debemos saber es cómo podemos llamar a una función de la *API* desde un programa desarrollado en *Visual Basic*. Cuando se instala *Visual Basic* en nuestro ordenador también se instala la aplicación *Visor de textos API*.

Antes de acceder a una clave o entrada del registro lo primero que debemos hacer es abrirla

Se trata de un programa sencillo que sirve para copiar las declaraciones de procedimientos de la *API* y pegarlos a nuestro código en *Visual Basic*. El *Visor de textos API* saca las declaraciones de un fichero de texto denominado **Win32api.txt**. El funcionamiento es bien sencillo. Seleccionado el tipo, la constante o la declaración que queremos incluir, lo copiamos y posteriormente lo pegamos en el código del programa.

APERTURA DE UNA CLAVE

Antes de acceder a una clave o entrada del registro lo primero que debemos hacer es abrirla. Esta es la finalidad de la función *RegOpenKeyEx* que viene declarada de la siguiente manera:

```
Declare Function RegOpenKeyEx Lib
    "advapi32.dll" Alias
    "RegOpenKeyExA" (ByVal hKey
    As Long, ByVal lpSubKey As
    String, ByVal ulOptions As
    Long, ByVal samDesired As
    Long, phkResult As Long) As
    Long
```

- *hkey*. Identificador de la clave principal que deseamos abrir. Normalmente será una constante que se corresponda con una de las claves principales que hemos descrito antes: *HKEY_CLASSES_ROOT*, *HKEY_CURRENT_USER*, *HKEY_LOCAL_MACHINE*, *HKEY_USERS*, *HKEY_CURRENT_CONFIG* o *HKEY_DYN_DATA*. Estas constantes deberán ser declaradas en nuestro código ya que son proporcionadas también por la propia *API*
- *lpSubKey*. Nombre de la clave que se desea abrir. Por ejemplo, en la clave *HKEY_CLASSES_ROOT\WinZip\shell\open\command* se encuentra la ruta donde se encuentra instalada la aplicación *Winzip* en nuestro sistema. Pues bien, el valor de *lpSubKey* para este caso sería la parte de esa clave correspondiente a *WinZip\shell\open\command*.
- *ulOptions*. Parámetro reservado (debe dejarse siempre con valor igual a 0).
- *samDesired*. Constante que identifica los permisos con los que se desea acceder a esa clave. Debe declararse, al igual que ocurre con las otras constantes. La tabla 1 muestra cuales son estas constantes y su significado.
- *phkResult*. Variable en donde se devuelve el indentificador de la rama abierta. Esta variable se utilizará posteriormente en el resto de las llamadas de la *API* cuando queremos modificar y/o leer elementos de esa clave.

Esta función devolverá el valor *ERROR_SUCCESS* en el caso de que la operación se haya llevado a cabo con éxito o bien otro valor si ha ocurrido un error.

CIERRE DE UNA CLAVE

Evidentemente también existe una llamada en la *API* para cerrar una clave que previamente hemos abierto.

```
Declare Function RegCloseKey Lib
    "advapi32.dll" (ByVal hKey As
    Long) As Long
```

- *hkey*. Identificador de la clave que deseamos abrir. Este identificador ha de corresponderse con el valor que devuelve la función *RegOpenKeyEx* a través de su parámetro *phkResult*.

NUESTRO PRIMER EJEMPLO

Hasta el momento no hemos visto gran cosa pero ya podemos realizar nuestro primer programa que acceda al registro de *Windows*. La figura 1 muestra el aspecto de la aplicación. Se trata simplemente de un programa que permite abrir entradas del registro. Si esto es posible aparecerá un mensaje informándonos que la operación ha concluido con éxito. En caso contrario recibiremos un mensaje de error. En primer lugar tenemos un módulo, **Registro.bas**, donde hemos guardado las declaraciones de constantes y procedimientos de la *API* que obtuvimos mediante el *Visor de textos API*. Este módulo es indispensable ya que sin él el programa sería incapaz de encontrar los procedimientos *RegOpenKeyEx* y *RegCloseKey*, o las constantes *HKEY_CLASSES_ROOT*, etc.

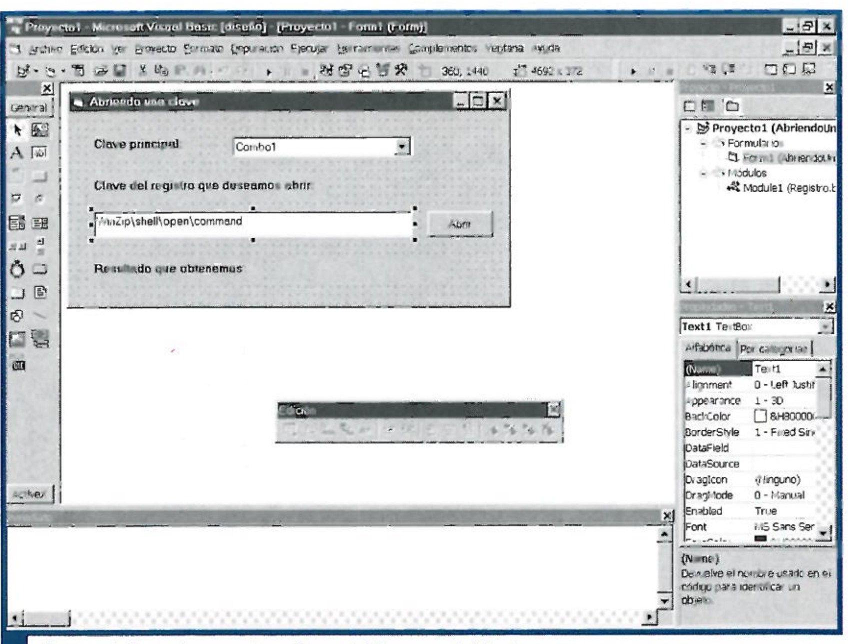


Figura1. Aplicación para abrir claves del Registro de Windows.

El funcionamiento es bastante sencillo. En la lista desplegable seleccionamos una de claves principales del registro. Posteriormente introducimos el resto de la entrada del registro en el área de texto y pulsamos el botón **Abrir** para comprobar si la clave se ha podido abrir con éxito. El listado 1 contiene el procedimiento que se ejecuta cuando se pulsa al botón **Abrir**.

LECTURA DE UN VALOR

Una clave del registro además de poder contener subclaves también puede albergar valores. Por ejemplo, en la clave *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion* se encuentran los valores **Registered Organization** y **Registered Owner** que contienen respectivamente el nombre y la compañía bajo los cuales se instaló *Windows*. *RegQueryValueEx* es el procedimiento de la *API* de *Windows* que permite conocer un valor de una clave ya dada.

```
Declare Function RegQueryValueEx
    Lib "advapi32.dll" Alias
    "RegQueryValueExA" (ByVal
    hKey As Long, ByVal
    lpValueName As String, ByVal
    lpReserved As Long, lpType As
    Long, lpData As Any, lpcbData
    As Long) As Long
```

- *hKey.* Identificador de la clave que contiene el valor que deseamos consultar. Este identificador ha de corresponderse con el valor que devuelve la función *RegOpenKeyEx* a través de su parámetro *phkResult*.
- *lpValueName.* Valor dentro de la clave.
- *lpReserved.* Parámetro reservado (Debe dejarse siempre con valor igual a **0**).
- *lpType.* Variable donde se almacena el tipo de la variable (la tabla 2 contiene los distintos tipos que pueden aparecer).
- *lpData.* Cadena donde se va a almacenar el contenido del valor.
- *lpcbData.* Número que expresa el tamaño de la cadena anterior.

Normalmente la lectura de un valor de una clave se suele hacer en dos fases. En primer lugar llamamos al procedimiento *RegQueryValueEx* sin pasarle ninguna cadena donde almacenar el valor. De esta forma en las variables correspondientes a *lpType* y a *lpcbData* se guardarán respectivamente el tipo del valor y el espacio requerido para almacenarlo.

```
lResultado =
    RegQueryValueExNULL(lIdClave,
    sNombreValor, 0&, lTipo, 0&,
    lTamanyoCadena)
```

Una vez que sabemos el espacio necesario creamos una variable del tipo que convenga y entonces leemos realmente el valor. El programa es similar al anterior pero además de abrir una entrada del registro consultamos el valor que el usuario haya indicado en la casilla de texto. *ERRORLECTURAREGISTRO* es una constante que hemos creado nosotros mismos y que será devuelta por la función siempre que se produzca un error en algún punto del procedimiento.

Normalmente la lectura de un valor de una clave suele realizarse en dos fases

De aquí en adelante siempre intentaremos encapsular las llamadas a la *API* en funciones. De este modo lograremos un mayor nivel de abstracción en nuestras aplicaciones y llevaremos únicamente a las funciones aquellos aspectos concretos de las llamadas (en vez de tener que realizar siempre el mismo tratamiento cada vez que queramos realizar una operación con el registro).

Por último, hay muchas entradas del registro para las cuales existe un valor predeterminado. En este caso podremos acceder a ese valor simplemente introduciendo la clave en la casilla correspondiente y dejando en

blanco la casilla de texto donde se introduce el nombre del valor.

El registro contiene información crítica para el sistema

Con esta pequeña aplicación que acabamos de realizar podemos tener acceso a todo tipo de información.

DESINSTALACIÓN DE LOS PROGRAMAS

Actualmente casi todos los programas que se instalan en *Windows* ofrecen la posibilidad de desinstalación. Esto lo vemos porque aparece una referencia en la pestaña *Instalar o desinstalar* dentro de **Inicio > Configuración > Panel de control > Agregar o quitar programas.** Internamente esto se traduce en una entrada en el registro de *Windows* en la clave:

```
HKEY_LOCAL_MACHINE\SOFTWARE\
    Microsoft\Windows\
    CurrentVersion\Uninstall
```

Es posible conocer datos con respecto a la desinstalación de los programas accediendo a determinados valores del registro. Utilizando nuestra aplicación podemos consultar el valor de *UninstallString* de la desinstalación de la aplicación *WinZip* consultando la clave:

```
HKEY_LOCAL_MACHINE\SOFTWARE\
    Microsoft\Windows\
    CurrentVersion\Uninstall\
    WinZip
```

El resultado que obtenemos es:

```
C:\ARCHIVOS DEPROGRAMA\WINZIP\
    WINZIP32.EXE /uninstall
```

Tabla 1. Constantes que identifican los permisos con los que abrimos una clave.

Constante	Tipo de acceso
KEY_CREATE_LINK	Creación de enlaces
KEY_CREATE_SUB_KEY	Creación de claves
KEY_ENUMERATE_SUB_KEYS	Enumeración de claves
KEY_EXECUTE	Lectura
KEY_QUERY_VALUE	Lectura de valores
KEY_SET_VALUE	Escritura de valores
KEY_NOTIFY	Notificación de cambios
KEY_READ	KEY_QUERY_VALUE + KEY_NOTIFY + KEY_ENUMERATE_SUB_KEYS
KEY_WRITE	KEY_QUERY_VALUE + KEY_CREATE_SUB_KEY
KEY_ALL_ACCESS	Cualquier acción

TIPOS DE ARCHIVOS DEL EXPLORADOR

Cuando abrimos el explorador de *Windows* y observamos los archivos de un determinado directorio, aparece para cada uno de ellos una pequeña descripción en la columna titulada *Tipo*. Esta descripción está almacenada en el Registro para cada extensión de archivo de la que el sistema tenga constancia. Así por ejemplo si consultamos el valor predeterminado de la clave:

```
HKEY_CLASSES_ROOT\.vbp
```

LAS CARPETAS ESPECIALES DE WINDOWS

En *Windows* existen una serie de carpetas o directorios especiales que tienen cierto significado particular para el sistema e incluso para algunas aplicaciones. Por ejemplo, en el directorio *C:\WINDOWS\SendTo* se encuentran los accesos directos que luego aparecen cuando seleccionamos un archivo, pulsamos en el botón derecho del ratón y elegimos la opción **Enviar a.** O la carpeta *C:\WINDOWS\Favoritos*, donde se guardan los *bookmarks* de *Internet Explorer*. Las rutas de todas estas carpetas están almacenadas en el registro de *Windows* en una serie de valores asociados a la clave:

```
HKEY_CURRENT_USER\Software\
    Microsoft\Windows\
    CurrentVersion\Explorer\
    Shell Folders
```

Algunos de estos valores son: *Cache* (donde se guardan temporalmente los archivos asociados a las páginas de *Internet* que hemos visitado utilizando *Internet Explorer*), *Cookies* (donde se guardan las *cookies* de las páginas *Web*), *Desktop* (el escritorio de *Windows*), *Fonts* (donde se almacenan las fuentes del sistema), etc. Consultados todos estos valores averiguaremos las rutas completas de estos directorios "especiales". En este artículo hemos visto como podemos leer ciertos datos de información del registro. En los siguientes seguiremos analizando el resto de las funciones de la *API* que permiten acceder al registro de *Windows* y, como no, continuaremos ofreciendo una perspectiva práctica del registro de forma que cada vez tengamos una idea más precisa de qué se guarda en el Registro, dónde se guarda y cómo afecta esto a nuestro sistema.

TRANSFERENCIA DE ARCHIVOS PUNTO A PUNTO

Enrique de la Lastra (enriquelastra@hotmail.com)

Si queremos conectar dos PC's para transferir archivos de uno a otro no es necesario que dispongamos de una tarjeta de red. Podemos emplear un cable que conecte los puertos serie de ambos y utilizar un programa que se encargue de la correcta comunicación de la información. Mediante este sistema se puede realizar transferencias de datos entre dos máquinas.

INTRODUCCIÓN

Un ordenador de sobremesa (ya sea un PC o un MAC) tiene una serie de puertos de comunicaciones que permiten conectarlo a otros dispositivos. Así, la impresora se conecta a nuestra máquina a través del puerto paralelo. El teclado se conecta a un puerto especial dispuesto a tal efecto (también el ratón si el ordenador dispone de un puerto específico). Además de estos puertos existen los puertos serie, donde podemos conectar el ratón, el módem, o incluso un cable para comunicar el ordenador con otra máquina.

En el presente y sucesivos artículos, iremos descubriendo cómo programar - mediante funciones traducidas de la API de Windows- el puerto serie para intercambiar, de forma segura y fiable, archivos y mensajes de texto entre dos PC's.

Lo más práctico es implementar cada elemento por separado

El enfoque que se va a seguir en este tipo de proyecto se puede orientar desde dos puntos de vista: el primero, implementar una aplicación que lleve a cabo todo el proceso de programación y control de la comunicación, al tiempo que muestre una interfaz de usuario adaptada al propósito de transmitir archivos y el segundo, y más práctico, implementar por separado cada elemento e integrar el conjunto en una aplicación específica al fin perseguido. Este segundo enfoque será el que sigamos.

El resultado de nuestro esfuerzo lo integraremos en un conjunto de tres componentes VCL (Visual Component Library). El primero de ellos, será el encargado de invocar a las funciones de comunicaciones de la API de Windows y manejar los errores generados. El segundo, y apoyándose en el primero, implementará los mecanismos de control de la comunicación evitando pérdidas y duplicados de información. El tercero y último, controlará la transmisión de archivos y mensajes, suministrando todas las

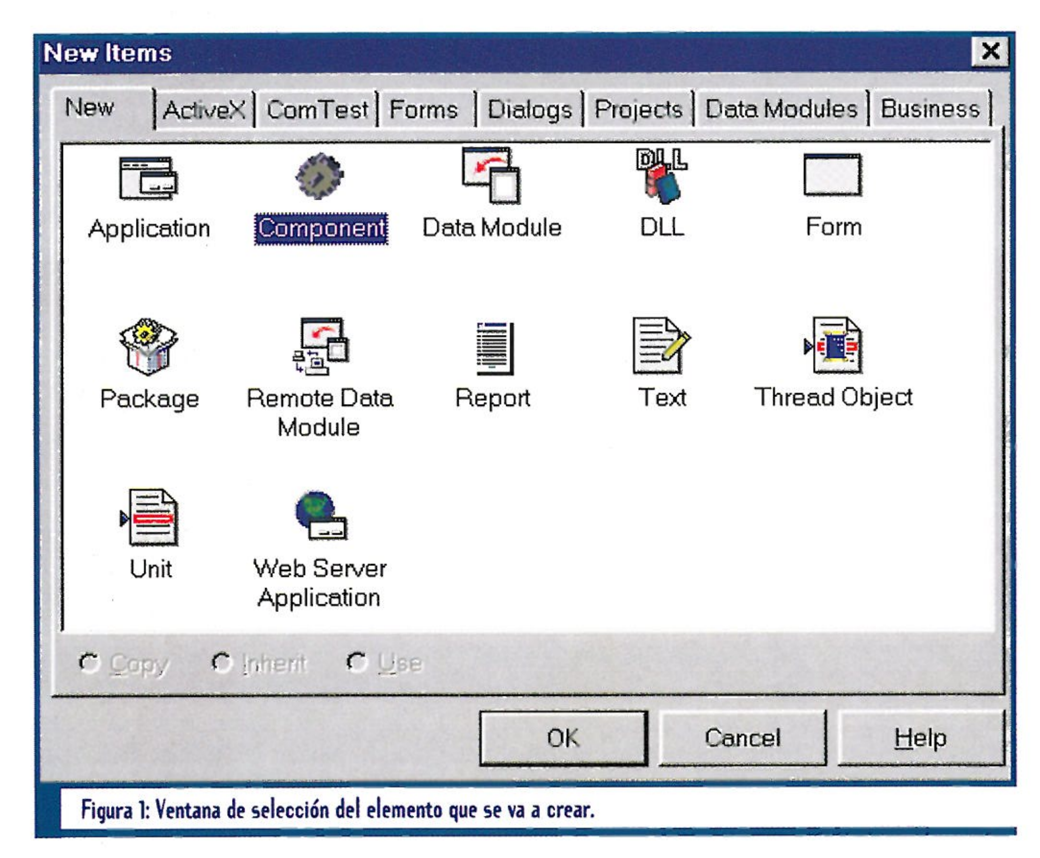


Figura 1: Ventana de selección del elemento que se va a crear.

funciones necesarias para realizar la transmisión de archivos de forma sencilla para el usuario (que en este caso será el programador que utilice el componente VCL).

Este último componente contendrá toda la funcionalidad necesaria para transmitir archivos y mensajes de forma fiable. Por tanto será de propósito general y se podrá utilizar en cualquier aplicación que persiga ese fin. En nuestro caso particular, programaremos una aplicación que haga uso del mismo para transmitir archivos entre dos PC's.

MODELO DE COMUNICACIONES

Las comunicaciones entre ordenadores suponen un proceso complejo bastante difícil de abordar en una sola aplicación. Para facilitar la problemática del diseño de los programas de comunicaciones la organización ISO (International Standardization Organization, Organización Internacional de Estandarización) promovió el desarrollo de un Modelo Funcional de Comunicaciones. Este modelo se denominó OSI cuyas siglas en inglés - Open Systems Interconection - significan Interconexión de Sistemas Abiertos. Con este modelo, la ISO definió cómo estructurar la implementación del *software* y del *hardware* necesarios para comunicar sistemas no propietarios (de ahí el término "sistemas abiertos").

El modelo OSI de la ISO (lo que parece un juego de palabras) estructura el proceso de comunicaciones de datos entre distintas máquinas en una serie de niveles funcionales, cada uno de los cuales lleva a cabo una serie de tareas diferenciadas y "únicas" (digámoslo entre comillas, ya que no es del todo cierto). Frecuentemente se utiliza la analogía de asemejar estos niveles a las capas de una cebolla. La razón es que cada uno de estos niveles realiza una función y oculta al resto de los niveles la forma en que la lleva a cabo. Se puede asimilar, por tanto, a una capa de una cebolla que no deja ver la capa que está debajo.

El modelo OSI estructura el proceso de comunicaciones en varios niveles funcionales

La principal utilidad de la estructuración de una comunicación en estos niveles lógicos es la capacidad de abstracción. Es decir, nosotros podemos desarrollar el software de un nivel abstrayéndonos de cómo el nivel inferior realiza sus funciones.

El conjunto de funciones que realiza un nivel, se denomina servicio. El servicio de un nivel será lo que, a la postre, va a utilizar de él su nivel superior. La forma en que ese nivel realiza el servicio queda ocultada para el nivel superior, consiguiéndose de esta forma que este último sólo se preocupe de las tareas que le atañen y se abstraiga de cómo el nivel inferior realiza las suyas.

Un servicio es el conjunto de funciones que ofrece un nivel al nivel superior

El principal defecto del modelo OSI es que es demasiado complejo y muchas veces repite tareas en varios niveles distintos. De hecho, los protocolos IP se desarrollaron al margen de

este modelo – antes incluso de que existiera el modelo OSI - y son en la práctica los protocolos que se han impuesto en el mundo de las comunicaciones. Por tanto, para nuestro desarrollo particular, seguiremos la filosofía de OSI pero sin ceñirnos estrictamente a sus especificaciones.

NUESTRO DISEÑO

El modelo OSI establece siete niveles y determina las funciones que realizará cada uno de ellos. Los cuatro primeros niveles están orientados a la comunicación de datos, mientras que los tres últimos están pensados desde el punto de vista de las aplicaciones de comunicaciones. La definición de los tres últimos niveles es muy poco clara y a efectos prácticos nosotros los consideraremos, en conjunto, como uno solo: el nivel de aplicación (tal y como sucede en los protocolos de Internet).

Seguiremos la filosofía OSI, pero sin ceñirnos estrictamente

Las funciones de los cuatro niveles más bajos, según OSI, son.

- Nivel 0: En realidad el nivel 0 no es un nivel como tal, sino el medio de transmisión físico. Puede ser un cable coaxial, el aire, una fibra óptica, etc. Su fin es el de transportar la señal en que se modulan los datos.
- Nivel 1: Éste es el primer nivel lógico. Su función es, por un lado, convertir los datos enviados (unos y ceros, en último término) en señales transportables por la línea y por otro lado, recuperar esas señales en el extremo receptor y de esas señales obtener los datos originales (otra vez unos y ceros). Es decir, debe transformar los datos binarios en señales eléctricas, por ejemplo un 1 en una señal de 5 Voltios y 1 microsegundo de duración, y un 0 en una señal de –5 Voltios e igual duración. De esas señales debe obtener en el receptor los "unos" y "ceros" enviados. También se ocupa de definir los conectores que utilizan los cables, las funciones de cada uno de los pines de esos conectores y las señales eléctricas que se envían por cada pin.
- Nivel 2: Trata de conseguir que la comunicación entre dos equipos conectados directamente sea fiable. Esto es, debe garantizar el diálogo entre dos máquinas y que lo que se recibe en una máquina sea igual a lo que envió la otra. Sus funciones serán: determinar quién inicia y quién termina la conexión; evitar que uno envíe datos si ya está el otro utilizando el medio de transmisión; detectar si ha habido errores (por ruido en la línea o interferencia con las adyacentes) y en caso afirmativo solicitar la retransmisión de los datos; detectar si hay información duplicada, etc.
- Nivel 3: Tiene la finalidad de conectar máquinas de distintas redes, independientemente de la tecnología de cada red. Es decir, debe poder comunicar dos máquinas que no estén conectadas directamente. Para ello debe definir una dirección única para cada máquina, y utilizar funciones de encaminamiento para que la información llegue desde cualquier origen a cualquier destino, pasando por cualquier número de redes.
- Nivel 4: Su función es la de conseguir una comunicación extremo a extremo fiable, ya que el nivel 3 no garantiza esta fiabilidad. Es decir, su función es, en cierto modo, semejante al nivel 2, pero extremo a extremo (ya que el nivel 2 debe garantizar una comunicación fiable, pero punto a punto).

En nuestro caso, el nivel 0 es un cable cruzado

En nuestro caso, el nivel 0 es un cable serie "cruzado" (como explicamos más adelante). El nivel 1, será el encargado de transmitir los datos al cable y de recuperarlos en el otro extremo. El conector será un RS-232. Este conector define, entre otros, un pin para enviar las señales de datos y otro para recibirlas. Teniendo en cuenta que al comunicar dos ordenadores mediante este tipo de conector, enfrentamos los pines, habrá que cruzar los cables de transmisión y recepción para que el pin de transmisión de un extremo se comunique con el de recepción del otro extremo. Por esta razón se conoce a este tipo de cable como "cable cruzado".

El nivel 1 también deberá convertir los datos del ordenador en señales eléctricas y transmitir esas señales. Para implementarlo utilizaremos las funciones de comunicación que incorpora la API de Windows (que de hecho, forman parte conceptualmente de ese nivel 1).

El nivel 2 se encargará de que los datos que transmitimos desde un ordenador sean iguales a los que recibimos en el otro ordenador. Para ello, dividiremos la información en tramas (que no son más que agrupaciones lógicas de bits, delimitadas por un principio y un fin, y con una serie de bits de control de la comunicación). Definiremos la estructura de esas tramas y los campos de control de las

mismas (tamaño de la trama, códigos de detección de errores, etc.).

Además, definiremos un protocolo que se encargue de coordinar la comunicación de forma que cada extremo sepa qué trama se está enviando o recibiendo y repita las tramas que el otro extremo haya recibido erróneamente.

En cuanto al nivel 3, como tiene por objetivo comunicar ordenadores en un entorno de red, y dado que en nuestro caso particular la comunicación es punto a punto entre dos PC's, resulta que para nuestra aplicación, este nivel está vacío (es decir, no realizaremos ninguna función en este nivel).

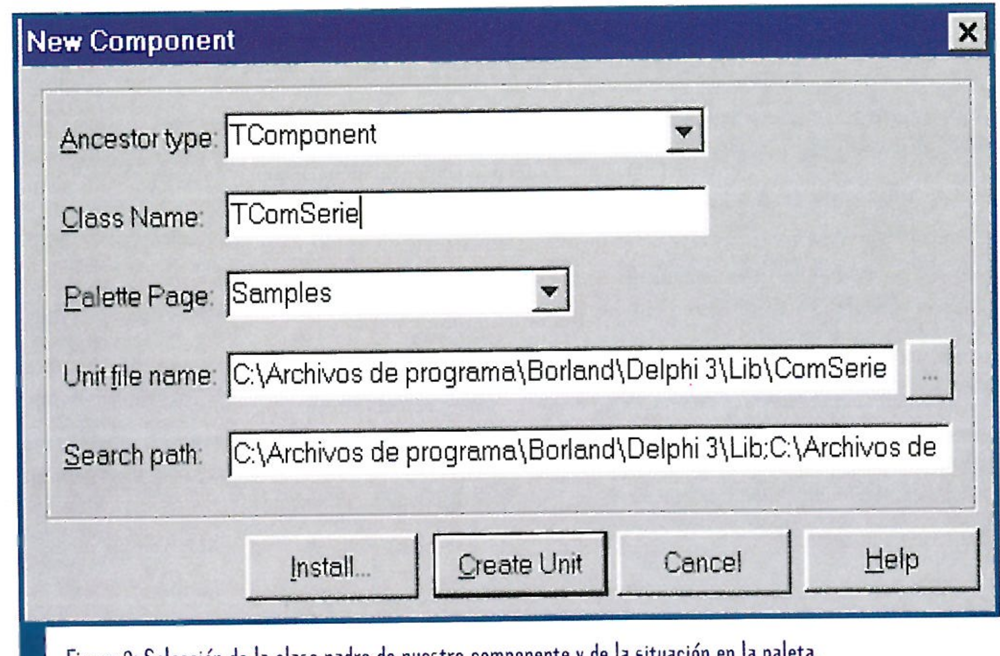


Figura 2: Selección de la clase padre de nuestro componente y de la situación en la paleta.

En una transmisión punto a punto, los niveles 3 y 4 del modelo OSI están vacíos

Igualmente, como el nivel 4 se encarga de que sea fiable la comunicación entre dos extremos de una red, y no es el caso, también estará vacío para nuestra aplicación particular.

Por tanto el diseño que seguiremos para implementar la transmisión de archivos por el puerto serie, quedará como sigue:

1. Un componente VCL llevará a cabo las funciones del nivel 1, encapsulando las funciones de la API de Windows y ofreciendo al nivel 2 unas rutinas de comunicaciones sencillas.
2. Otro componente VCL, apoyándose en el anterior (de hecho será un componente heredado), implementará el nivel 2 del modelo explicado. Por tanto, definirá los procedimientos necesarios para la transmisión de las tramas y para el control del protocolo a nivel 2 (detección de errores, control de flujo, etc.).

Separaremos el desarrollo en tres componentes VCL heredados cada uno del anterior

3. Un último componente VCL, heredado del anterior, ofrecerá las funciones necesarias para la comunicación de archivos, encargándose de trocearlos en partes que se adapten al tamaño de las tramas, y controlando la correcta transmisión de los archivos completos. Aquí se podrán añadir cuantas funciones quiera el programador, como pueden ser: verificación del nombre de los archivos y de su existencia; establecimiento de marcas que permitan recuperar la transferencia en el momento en el que se encontraba sin necesidad de reiniciar la transferencia completa; transmisión de mensajes de texto con ficheros anexados, etc.
4. Por último, una aplicación hará uso del servicio suministrado por el componente anterior, implementando un programa práctico que sirva tanto para transmitir ficheros como mensajes escritos desde el teclado.

PRIMER COMPONENTE

Vamos pues a comenzar con el primer componente VCL de los tres que vamos a desarrollar. Para su implementación haremos uso de las funciones de comunicaciones de la API de Windows, las cuales cubren todo lo necesario para establecer una comunicación a través del puerto serie.

El componente lo heredaremos de *TComponent*, ya que aunque no

necesita ser visual (por tanto no necesitamos heredarlo de *TControl*), si queremos poder editar sus futuras propiedades y eventos desde el *Object Inspector* de Delphi.

Para crear el componente seleccionaremos el menú *File/New...* (en Delphi 3.0) y en la ventana que aparece (ver figura 1) seleccionaremos *Component* y pulsaremos el botón OK.

Aparecerá una ventana como la mostrada en la figura 2, en la que elegiremos como *Ancestor Type* (Clase Padre) *TComponent* y escribiremos el nombre de la clase (aquí hemos elegido el nombre: *TComSerie*). El resto de las entradas del diálogo se pueden dejar con los valores por defecto. Ya sólo resta pulsar OK y Delphi se encargará de crear el esqueleto básico de la nueva clase *TComSerie* que se creará en una unit (unidad) denominada *ComSerie*.

ABRIR UN PUERTO DE COMUNICACIONES

La API de Windows suministra todas las funciones necesarias para programar cualquier recurso de comunicaciones, con la única pega que son mucho menos sencillas de utilizar que las de Delphi. En este sentido, Delphi no aporta ninguna facilidad, tan sólo se limita a realizar una traducción de las funciones de comunicaciones a otras equivalentes de *Object Pascal*.

Para utilizar el puerto serie del ordenador, debemos primero abrirlo, para lo cual hay que crear un *handle* (manejador) a ese puerto. Este *handle* es un puntero al puerto y se asocia unívocamente a él, de forma que una vez creado lo utilizaremos siempre en todas las funciones de la API de *Windows* que utilicen el puerto.

La función de la API que crea este *handle* se denomina *CreateFile*. En realidad esta función es de propósito general y no es específica para crear recursos de comunicaciones. Su declaración es la siguiente:

```
function CreateFile(
  lpFileName: PChar;
  dwDesiredAccess, dwShareMode:
    Integer;
  lpSecurityAttributes: PSe-
    curityAttributes;
  dwCreationDisposition, dwFlag-
    sAndAttributes: DWORD;
  hTemplateFile: THandle):
    THandle;
);
```

De estos parámetros - y debido al propósito no específico de la función - hay tres que deben asumir unos valores determinados para conseguir que la creación del *Handle* se dirija a un recurso de comunicaciones (en nuestro caso, el puerto serie):

- dwShareMode: debe valer 0.
- dwCreate: se le debe asignar el flag OPEN_EXISTING.
- hTemplateFile: debe ser nil.

Para abrir un puerto de comunicaciones hay que crear un handle al puerto

El primer parámetro *lpFileName* indica el recurso que abrir. Para nuestra aplicación particular el recurso será un puerto serie (COM1 o COM2), y por tanto pasaremos el string COM1 o COM2. Además se debe especificar, a través del parámetro *dwDesiredAccess* si el puerto de comunicaciones se va a utilizar como entrada de datos (GENERIC_READ), como salida (GENERIC_WRITE) o como entrada y salida (GENERIC_READ or GENERIC_WRITE). El parámetro *lpSecurityAttributes* debe permitir que el *handle* pueda ser heredado por procesos hijos (en una aplicación multithreading). El tipo de este parámetro es:

```
PSecurityAttributes = ^TSecurit-
    yAttributes;
TSecurityAttributes = record
  nLength: DWORD;
  lpSecurityDescriptor: Pointer;
  bInheritHandle: BOOL;
end;
```

La variable *bInheritHandle* es la que indica si el *handle* puede ser heredado (*nLength* debe indicar el tamaño de la estructura y fijando *lpSecurityDescriptor* a *nil* se toma la política de herencia por defecto).

Como podemos observar, la función devuelve una variable de tipo *THandle*, que será el *handle* creado al dispositivo. Si al intentar abrir el puerto, éste ya se encuentra abierto, la función falla y devuelve 0 (se generará un error que luego podremos comprobar con otra función de la API específica para el manejo de errores, *GetLastError*, cuyo funcionamiento veremos más adelante). En caso de que el puerto no esté abierto, se inicializa automáticamente con los valores asignados la vez anterior (si fue abierto con anterioridad), o los valores por defecto, si es la primera vez que se abre el puerto.

CONFIGURACIÓN DEL PUERTO SERIE

Uno de los valores que se conserva al abrir un puerto es el de la estructura DCB (Device Control Block, Bloque de Control del Dispositivo), que contiene todas las opciones de configuración del puerto (ver listado 1) tasa de bits, bits de datos y bits

de parada, bits de paridad, control de flujo, etc.

La estructura DCB contiene las variables que permiten configurar un puerto de comunicaciones

Para obtener los valores de esta estructura disponemos de una función específica de la API denominada *GetCommState*. Esta función sólo tiene dos parámetros: el *handle* al recurso de comunicaciones y una variable de tipo TDCB. Pasando como parámetro el *handle* del puerto (que nos devuelve la función *CreateFile*), rellena la estructura (ver listado 1) con los valores de configuración actuales del dispositivo.

La utilidad principal de esta función es la de asignar valores válidos a todas las variables de la estructura TDCB, ya que normalmente, cuando queramos cambiar o asignar los parámetros de la comunicación sólo vamos a modificar algunas variables y no todas las de la estructura.

La función definida en la API para este propósito es *SetCommState* y tiene los mismos parámetros que la anterior (aunque, lógicamente el parámetro que apunta a la estructura DCB es ahora una entrada de datos). Esta función se encarga de reconfigurar el puerto serie de acuerdo a los nuevos valores de las variables de DCB. Sin embargo, no permite configurar los buffers de entrada y salida de datos. Para este fin, existe una nueva función denominada *SetupComm* (que en realidad es opcional, ya que, si no se invoca, se asignan a ambos *buffers* unos tamaños por defecto).

La función SetCommState cambia los valores de la estructura TDCB

SetupComm tiene tres parámetros: el primero, como siempre, es el *handle* al puerto, y los otros dos son los tamaños recomendados para los *buffers* de entrada y salida. Los tamaños son recomendados porque, en la práctica, la función *SetupComm* los asigna según le parece conveniente siguiendo unos criterios de rendimiento.

En cuanto al resultado, si realiza su tarea con éxito devuelve TRUE, mientras que en caso de error devuelve FALSE (en este último caso, podremos conocer el error generado llamando a la función *GetLastError,* que como ya hemos comentado, veremos más adelante).

La función *SetupComm,* antes de asignar el nuevo tamaño a los buffers de entrada y salida, termina todas las operaciones de lectura y escritura del puerto - aunque

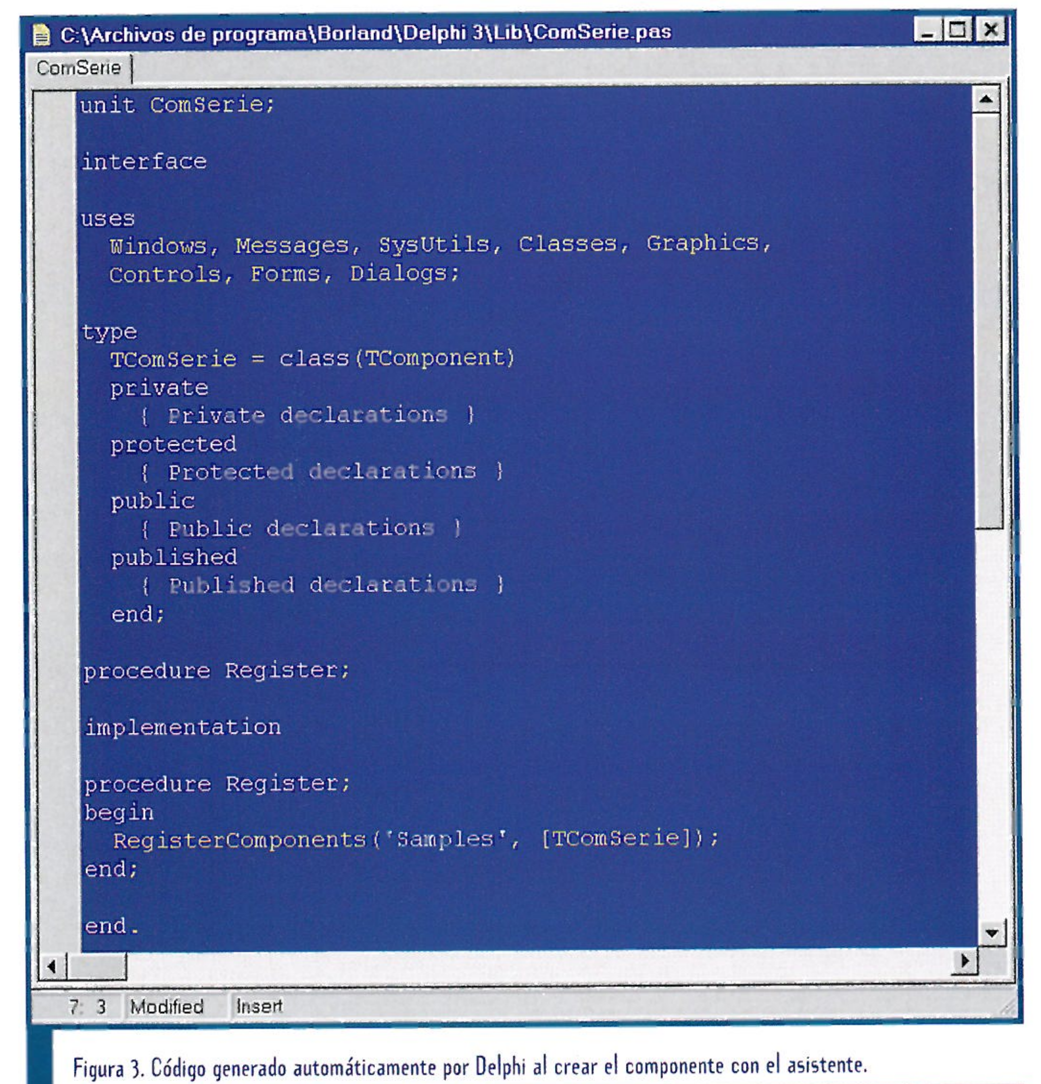


Figura 3. Código generado automáticamente por Delphi al crear el componente con el asistente.

Listado 1: Estructura _DCB (Device Control Block) de la API y traducción de Delphi (TDCB).

```
typedef struct _DCB {
    DWORD DCBlength;      { Tamaño de la estructura (sizeof(DCB))}
    DWORD BaudRate; { Tasa de bits actual }
    DWORD fBinary: 1;     { Modo binario, sin comprobación de EOF }
    DWORD fParity: 1;     { Permite la comprobación de paridad }
    DWORD fOutxCtsFlow: 1;     { Control de flujo de salida Clear-To-Send }
    DWORD fOutxDsrFlow: 1;     { Control de flujo de salida Data-Set-Ready }
    DWORD fDtrControl: 2;      { Control de flujo Data-Terminal-Ready }
    DWORD fRtsControl: 2;      { Control de flujo Request-To-Send }
    DWORD fDsrSensitivity: 1; { Sensibilidad Data-Set-Ready }
    DWORD fTXContinueOnXoff: 1;     { Termina transmsión cuando llega XOFF }
    DWORD fOutX: 1; { Control de flujo de salida XON/XOFF }
    DWORD fInX: 1;  { Control de flujo de entrada XON/XOFF }
    DWORD fErrorChar: 1;       { Permite la sustitución de los bytes erróneos por el byte ErrorChar
}
    DWORD fNull: 1; { Permite eliminar los bytes nulos }
    DWORD fAbortOnError: 1;    { Aborta la lectura y escritura si hay error }
    DWORD fDummy2: 17;    { Reservado }
    WORD wReserved; { Reservado }
    WORD XonLim;    { Número de bytes en el buffer de entrada para transmitir XON }
    WORD XoffLim;   { Número de bytes en el buffer de salida para transmitir XOFF }
    BYTE ByteSize;  { Número de bits por byte (4-8) }
    BYTE Parity;    { Tipo de paridad (0-4=ninguna, impar, par, marca, espacio) }
    BYTE StopBits;  { Bits de parada (0,1,2 = 1bit, 1.5 bits, 2 bits) }
    char XonChar;   { Valor del carácter XON, para transmisión y para recepción }
    char XoffChar;  { Valor del carácter XOFF, para transmisión y para recepción }
    char ErrorChar; { Carácter de sustitución en caso de error }
    char EofChar;   { Carácter EOF }
    char EvtChar;   { Carácter para indicar un evento }
    WORD wReserved1;      { Reservado }
} DCB;

type
  TDCB = packed record
    DCBlength: DWORD;
    BaudRate: DWORD;
    Flags: Longint;
    wReserved: Word;
    XonLim: Word;
    XoffLim: Word;
    ByteSize: Byte;
    Parity: Byte;
    StopBits: Byte;
    XonChar: CHAR;
    XoffChar: CHAR;
    ErrorChar: CHAR;
    EofChar: CHAR;
    EvtChar: CHAR;
    wReserved1: Word;
  end;
  PDCB = ^TDCB;
```

no hayan terminado -, descarta los bytes no leídos y vacía los dos *buffers*.

Vistas estas funciones, la apertura del puerto serie implicará una serie de llamadas a las funciones de la API, que tendrán, más o menos, la siguiente forma:

```
TComSerie = TComponent
private
  ComHandle: THandle;
  DCB: TDCB
  InBufSize: DWORD;
  OutBufSize: DWORD;

...

implementation

...
procedure TComSerie.AbrirPuerto;
  ComHandle :=
    CreateFile("COM1",
    GENERIC_READ or
    GENERIC_WRITE,
      0, atributos_de_seguridad
    , OPEN_EXISTING,
    flags_y_atributos, 0) ;

  if ComHandle > 0 then
    GetCommState (ComHandle,
    DCB);
    with DCB do begin
      DCBlength:=
    valor_DCBlength;
      BaudRate:=  valor_BaudRa-
    te;
      Flags:=  valor_Flags;
      XonLim:=  valor_XonLim;
      XoffLim:=  valor_XoffLim;
      ByteSize:=  valor_ByteSi-
    ze;
      Parity:=  valor_Parity;
      StopBits:=  valor_Stop-
    Bits;
      XonChar:=  valor_XonChar;
      XoffChar:=  valor_Xoff-
    Char;
      ErrorChar:=  valor_Error-
    Char;
      EofChar:=  valor_EofChar;
      EvtChar:=  valor_EvtChar;
    end;
  SetCommState (ComHandle, DCB);
  SetupComm (ComHandle, InBufSi-
    ze, OutBufSize);
end;
```

OBTENER LOS ERRORES PRODUCIDOS

Si al invocar cualquier función de la API de comunicaciones se produce un error y queremos saber cuál es, utilizaremos la función *GetLastError*. Esta función, que se puede invocar en cualquier momento, devuelve el código del último error generado. El valor de este código, que será mayor que cero, se puede fijar con la función *SetLastError*.

La función *GetLastError* puede ser muy útil para depurar nuestro programa, ya que permite conocer, en el momento en que falla la aplicación, cuál fue la causa del fallo. Por ejemplo, si al llamar a la función *CreateFile* intentamos abrir un recurso de comunicaciones que ya se encontraba abierto, una llamada a la función *GetLastError*, devolverá el código ERROR_ALREADY_EXISTS. Si la función falla por otro motivo, la función *GetLastError* devolverá el código INVALID_HANDLE_VALUE.



Figura 4. Capas de comunicaciones del modelo OSI.SeleccionaFuente.

CONCLUSIÓN

Hemos estructurado el proceso de creación de una aplicación de transmisión de archivos por el puerto serie, en una serie de componentes VCL, heredados cada uno del anterior, que permiten simplificar la programación del conjunto gracias al método “divide y vencerás”.

Además hemos dado los primeros pasos en la creación del primero de estos componentes, estudiando las funciones de la API de Windows que nos interesan y adaptándolas a nuestro desarrollo.

Una vez creado el primer componente nos queda el protocolo de comunicaciones

En el próximo artículo, terminaremos este primer componente, que permitirá transmitir a través del puerto serie cualquier tipo de información de datos. En los sucesivos artículos, haremos uso de este componente creado para desarrollar otro heredado del anterior, que implemente un protocolo de comunicaciones en el nivel 2 del modelo OSI.

DIRECTX 6.1 (III)

Constantino Sánchez Ballesteros (constantino@nexo.es)

Direct3D es la parte de DirectX se encarga del proceso gráfico en 3 dimensiones. Para ello aprovecha las últimas tecnologías para representar con la mayor fidelidad posible los entornos virtuales.

DIRECT3D

La creación de un *device* (dispositivo), construcción de una escena y animación de la misma son conceptos fundamentales de la API *Direct3D Modo Retenido*. El ejemplo que vamos a desglosar en este artículo demuestra esos conceptos mediante la creación de un *device* por defecto, carga de un objeto 3D a partir de un fichero, asignación de color y luces en la escena y rotación del objeto. En el proceso de construcción de la escena, el ejemplo utiliza un *meshbuilder* (constructor de mallas 3D), *frames* (para albergar objetos y luces), luces y un material. El ejemplo también creará un objeto *DirectDrawClipper* (para la representación de la escena en una ventana), un *Viewport* (donde se construye la escena que será visualizada) y una cámara.

El ejemplo viene contenido en un solo archivo de tipo cpp. No utiliza *headers* separados, excepto los utilizados para el manejo de *DirectX*. Tampoco se crean menús, iconos ni otros recursos típicos de aplicaciones *Win32*. En el ejemplo se utiliza una "tetera" como objeto 3D (*Teapot.x*) extraída del *SDK* de *DirectX* para visualizarla en la escena.

DEPENDENCIAS

A continuación se detallan las dependencias necesarias para la correcta creación y compilación del proyecto dentro del entorno Visual C++:

- Enlaces a las librerías *DDraw.lib* y *D3DRM.lib*.
- Carga de la tetera a partir del archivo *Teapot.x*, tal y como se describió en la sección anterior.
- Inicialización de *GUID's*; las aplicaciones de *Direct3D* deben contener la siguiente definición al principio del código:

```
#define INITGUID
```

Inclusión de los siguientes *headers*:

```
#include <windows.h>
    // header estándar de
    Windows
#include <direct.h>
    // DirectDraw: definiciones
#include <d3drmwin.h>
    // Direct3D Modo Retenido:
    definiciones
```

DECLARACIONES GLOBALES

Vamos a comenzar la creación del programa definiendo algunas constantes y variables globales que nos ayuden a organizar la información. La creación de la macro *RELEASE* se encargará de liberar un objeto creado sólo si éste no es *NULL* y se asegura que un objeto sea inicializado como *NULL* después de haber sido liberado. Esta práctica elimina el problema de intentar liberar un objeto que lo ha sido previamente, el cual puede causar resultados indeseables, como errores de protección.

```
// Macro para liberar un objeto
#define RELEASE(x) if (x !=
    NULL) {x->Release(); x =
    NULL;
}
```

La macro *DISPLAYMSG* utiliza la función de *Windows MessageBox* para visualizar información al usuario.

```
// Macro para visualizar un cua-
     dro de mensajes conteniendo
     la cadena asignada.
#define DISPLAYMSG(x)
     MessageBox(NULL, x, "Ejemplo
     de D3DRM ", MB_OK);
```

Las variables globales representan objetos clave, como los objetos *Direct3D Modo Retenido* y *DirectDrawClipper*. La estructura *myglobs* colecciona el *device*, *viewport*, la escena e información de la cámara, así como información sobre si la aplicación está minimizada cuando los objetos han sido inicializados.

```
// Variables globales
LPDIRECT3DRM3 lpD3DRM = NULL;
// objeto Direct3DRM
LPDIRECTDRAWCLIPPER lpDDClipper
     = NULL;
// objeto DirectDrawClipper

// estructura global
struct _myglobs {
// Direct3DRM Device.
LPDIRECT3DRMDEVICE3 dev;

// Direct3DRM Viewport: a través
     del cual se ve la escena
LPDIRECT3DRMVIEWPORT2 view;

// Frame principal que contiene
     los demás Frames que se
     creen
LPDIRECT3DRMFRAME3 scene;

// Frame que describe el punto
     de vista del observador
LPDIRECT3DRMFRAME3 camera;

// Aplicación está minimizada
BOOL bMinimized;

// Todos los objetos de Direct3D
     Modo Retenido han sido ini-
     cializados
BOOL bInitialized;
} myglobs;
```

SUMARIO DE FUNCIONES

El ejemplo contiene las siguientes funciones:

- *BuildScene*: crea la escena que será renderizada. Carga un objeto 3D y agrega un color al mismo, además de insertar luces en la escena. *BuildScene* también establece la posición de objetos y cómo deberían ser rotados. *CreateObjects*, que inicializa las variables globales y crea devices por defecto, frames. *Viewports* utiliza esta función.
- *RenderScene*: dibuja la escena que *BuildScene* creó. Cada vez que *WinMain* utiliza esta función, *RenderScene* mueve la escena, permitiendo la animación del objeto 3D.

Es necesario especificar un device para determinar quién gestionará la representación gráfica, hardware 3D ó software

- *AddMediaPath*: da una ruta a las texturas y archivos .x requeridos por la aplicación mediante su búsqueda en el Registro.
- *CreateObjets*: inicializa variables globales y crea un *device* por defecto, *frame* de cámara, *frame* de escena y *Viewport*. *CreateObjects* utiliza *BuildScene* para crear la escena que será renderizada. *InitApp* utiliza *CreateObjects*.
- *InitApp*: crea la ventana principal que utilizará el programa. Utiliza *CreateObjects* para inicializar los objetos globales.
- *WinMain*: establece el bucle principal de la aplicación. Utiliza *InitApp* para inicializar la aplicación y *RenderScene* para dibujar la escena.
- *WindowProc*: gestiona los mensajes de la aplicación. Pone énfasis para determinar cuándo la aplicación está minimizada o destruída.

FUNCIÓN BUILDSCENE

La función *BuildScene* crea la escena que será renderizada. Todas las aplicaciones de *Direct3D Modo Retenido* deben crear una escena, pero el código requerido depende de cada escena en particular. Típicamente, las aplicaciones crean frames y luces, agregan efectos visuales a la escena y establecen la posición de los objetos. A continuación se muestran los pasos utilizados para la construcción de la escena:

1. Carga un objeto *3D* a partir de un archivo mediante los métodos *CreateMeshBuilder* y *Load*.
2. Crear un frame hijo en la escena utilizando el método *CreateFrame*.
3. Agregamos al *frame* hijo el objeto *3D* cargado anteriormente mediante el método *AddVisual*:
4. Establecemos la posición de la cámara en la escena mediante *SetPosition*.
5. Establecemos la rotación del *frame* hijo utilizando *SetRotation*.
6. Creamos un *frame* para una luz que será hijo de la escena. Utilizaremos el método *CreateFrame* descrito anteriormente.
7. Posicionamos el *frame* de luz en la escena mediante *SetPosition*.

8. Creamos un brillo (luz tipo punto paralelo (*parallel point*) y lo agregamos al frame de luz utilizando *CreateLightRGB* y *AddLight*.
9. Creamos un ambiente (luz ambiente) y lo agregamos a la escena con los métodos del punto anterior:

La asignación de materiales y texturas a los objetos 3D acercan los entornos virtuales a una representación real

10. Creamos un material y asignamos los reflejos mediante *CreateMaterial* y *SetMaterial*.
11. Establecemos el color del objeto *3D* utilizando el método *SetColorRGB*.
12. Liberamos todo mediante la macro *Release*.

CÓDIGO FUENTE DE BUILDSCENE

A continuación se presenta el código fuente completo de la función BuilScene comentada en el apartado anterior. Puesto que antes se han destacado los pasos utilizados para la creación de la escena, es el momento de llevarlo todo a la práctica.

```
// Creamos la escena que será
     renderizada
BOOL BuildScene( LPDIRECT3DRM3
     lpD3DRM,
      LPDIRECT3DRMDEVICE3 dev,
     LPDIRECT3DRMFRAME3 scene,
      LPDIRECT3DRMFRAME3 camera )
     {
     LPDIRECT3DRMFRAME3 lights =
     NULL;
     LPDIRECT3DRMMESHBUILDER3
     meshbuilder = NULL;
     LPDIRECT3DRMFRAME3
     childframe = NULL;
     LPDIRECT3DRMLIGHT light1 =
     NULL;
     LPDIRECT3DRMLIGHT light2 =
     NULL;
     LPDIRECT3DRMMATERIAL2 mat =
     NULL;
     HRESULT rval;

// Cargamos un objeto 3D a par-
     tir de un archivo
if (FAILED(lpD3DRM->CreateMesh-
     Builder(&meshbuilder)))
     goto generic_error;

rval = meshbuilder->Load("tea-
     pot.x", NULL,
        D3DRMLOAD_FROMFILE, NULL,
     NULL);
if (FAILED(rval)) {
DISPLAYMSG("No se pudo cargar el
     archivo .X");
         goto ret_with_error;
}

// Creamos un frame hijo dentro
     de la escena
if (FAILED(lpD3DRM-
     >CreateFrame(scene, &child-
     frame)))
     goto generic_error;

// Insertamos el objeto 3D den-
     tro del frame hijo
if (FAILED(childframe->AddVisual(
     (LPDIRECT3DRMVISUAL) mesh-
     builder)))
     goto generic_error;

       // Establecemos la posi-
     ción de la cámara. Los obje-
     tos con el mismo valor x/y
     de la
       // cámara aparecerán justo
     enfrente. Los valores Z
     negativos son más lejanos,
     haciendo
       // que el objeto aparezca
     más pequeño tanto en cuanto
     incremente este valor
       // negativo.
rval = camera-
     >SetPosition(scene,
     D3DVAL(0), D3DVAL(0), -
     D3DVAL(7));
if (FAILED(rval)) {
     DISPLAYMSG("No se pudo posi-
     cionar la cámara en el
     frame. ");
     goto ret_with_error;
}

// Rotamos el frame hijo sobre
     el eje Y (para x/z el valor
     será igual a cero)
```



Figura 1: El mapeado de texturas permite crear efectos realmente sorprendentes como el de la imagen. En este caso se ha utilizado el Environment Mapping.

```
// y utilizará un ángulo rota-
    cional lento.
if (FAILED(childframe->SetRota-
    tion(scene, D3DVAL(0),
    D3DVAL(1), D3DVAL(0),
    D3DVAL(0.03))))
// ángulo  goto generic_error;

// Inicializamos las luces en la
    escena, creando un frame que
// es hijo del frame de escena.
if (FAILED(lpD3DRM-
    >CreateFrame(scene,
    &lights)))
    goto generic_error;

// Posiciona el frame de luz en
    la escena. Esta luz llega
    desde la derecha
// de la cámara. Tiene un valor
    diferente de X, pero los
    mismos valores
// de Y/Z que los utilizados
    para la posición de la cáma-
    ra.
if (FAILED(lights-
    >SetPosition(scene,

    D3DVAL(5), D3DVAL(0), -
    D3DVAL(7))))
    goto generic_error;

// Crea un brillo, luz de punto
    paralelo, y lo agrega al
    frame de luz
// Los valores de color deben
    estar en el rango 0.0 (oscu-
    ridad) a 1.0 (brillo).
if (FAILED(lpD3DRM->Create-
    LightRGB(
    D3DRMLIGHT_PARALLELPOINT,
    D3DVAL(1.0), D3DVAL(0.8),
            D3DVAL(0.9),
    &light1)))
goto generic_error;

if (FAILED(lights-
    >AddLight(light1)))
    goto generic_error;

// Creamos oscuridad, luz
    ambiente, y la agregamos al
    frame de escena,
// aplicándola a toda la escena.
    La luz ambiente llega de
    todas las
// direcciones...
if (FAILED(lpD3DRM->Create-
    LightRGB(D3DRMLIGHT_AMBIENT,
    D3DVAL(0.1), D3DVAL(0.1),
    D3DVAL(0.1), &light2)))
    goto generic_error;

if (FAILED(scene-
    >AddLight(light2)))
    goto generic_error;

// Crea un material, estable-
    ciendo el reflejo (5.0 es
    metálico,
// valores mayores son más plás-
    ticos) sobre el objeto 3D.
if (FAILED(lpD3DRM->CreateMate-
    rial(D3DVAL(10.0), &mat)))
    goto generic_error;

if (FAILED(meshbuilder->SetMate-
    rial(mat)))
    goto generic_error;

// Establecemos el color del
    objeto 3D (verde brillante
    en nuestro caso).
if (FAILED(meshbuilder->SetCo-
    lorRGB(D3DVAL(0.0),     // red
            D3DVAL(0.7),    //
    green
            D3DVAL(0.0))))  //
    blue
    goto generic_error;

// Liberamos objetos.
RELEASE(childframe);
RELEASE(lights);
RELEASE(meshbuilder);
RELEASE(light1);
RELEASE(light2);
RELEASE(mat);
return TRUE;

    generic_error:
DISPLAYMSG("Ocurrió un fallo
    mientras se construía la
    escena.");
    ret_with_error:
RELEASE(childframe);
RELEASE(lights);
RELEASE(meshbuilder);
RELEASE(light1);
RELEASE(light2);
RELEASE(mat);
return FALSE;
    }
```

FUNCIÓN RENDERSCENE

RenderScene dibuja la escena en la pantalla que *BuildScene* creó mediante la utilización de los siguientes pasos:

1. Mueve la escena utilizando el método *Move*.
2. Borra el *Viewport* utilizando *Clear*.
3. Renderiza la escena utilizando *Render*.
4. Actualiza la ventana utilizando *Update*.

CÓDIGO FUENTE DE RENDERSCENE

Después de ver los pasos necesarios para implementar la función *RenderScene*, pasamos a los hechos creando el código fuente correspondiente.

```
// Borramos el Viewport, Rende-
    rizamos el siguiente frame y
    actualizamos la ventana.
static BOOL RenderScene() {
HRESULT rval;

// Movemos la escena.
rval = myglobs.scene->Move
    (D3DVAL(1.0));
if (FAILED(rval)) {
    DISPLAYMSG("Fallo en anima-
    ción de escena.");
    return FALSE;
}

// Borramos el Viewport
rval = myglobs.view->Clear();
if (FAILED(rval)) {
```

```
        DISPLAYMSG("Fallo al borrar
        el Viewport.");
        return FALSE;
}

// Renderizamos la escena al
        Viewport.
rval = myglobs.view->Render
        (myglobs.scene);
if (FAILED(rval)) {
        DISPLAYMSG("Fallo en el Ren-
        derizado.");
        return FALSE;
}

// Actualizamos la ventana.
rval = myglobs.dev->Update();
if (FAILED(rval)) {
DISPLAYMSG("Fallo en la actualiza-
ción del dispositivo.");
return FALSE;
}
return TRUE;
}
```

FUNCIÓN ADDMEDIAPATH

AddMediaPath crea una ruta para las texturas y los archivos .x requeridos por la aplicación. La función busca en el Registro del sistema la localización de los ejemplos de *DirectX* para poder efectuar la carga del archivo *Teapot.x* utilizado para su renderización.

CÓDIGO FUENTE DE ADDMEDIAPATH

```
// Miramos en el registro de
        sistema para determinar la
        ruta de archivos multimedia
// y agregamos esa ruta al final
        de la ruta de búsqueda del
        fichero.
VOID AddMediaPath(LPDIRECT3DRM3
        pD3DRM) {
HKEY    key;
LONG    result;
TCHAR   strPath[512];
DWORD   type, size = 512;

// Abrimos el registro
result =
RegOpenKeyEx(HKEY_LOCAL_MACHINE,
        "Software\\Microsoft\\
        DirectX", 0, KEY_READ,
        &key);
if (ERROR_SUCCESS != result)
    return;

// Buscamos el valor de registro
        deseado y cerramos el regis-
        tro.
result = RegQueryValueEx(key,
        "DX6SDK Samples Path", NULL,
        &type,(BYTE*)strPath,
        &size);
RegCloseKey(key);

if(ERROR_SUCCESS != result)
        return;

strcat(strPath,"\\D3DRM\\Media");

pD3DRM->AddSearchPath(strPath);

return;
}
```

FUNCIÓN CREATEOBJECTS

CreateObjects inicializa las variables globales del ejemplo y crea objetos. Después de inicializar todas las variables globales, *CreateObjects* efectúa los pasos necesarios para crear un *device*, asociarlo con la ventana principal del programa y crear el frame principal de la escena, el frame de cámara y el *Viewport* de la siguiente forma:

1. Utilizamos la función de *DirectDraw* de *DirectDrawCreateClipper* para crear un objeto *DirectDrawClipper.*
2. Usamos el método *SetHWnd* de *DirectDrawClipper* para asociar la ventana de la aplicación con el objeto *DirectDrawClipper.* Este objeto limita las operaciones de dibujo a un área determinada, en este caso, la ventana de la aplicación.
3. Utilizamos la función *Direct3DRMCreate* para crear un objeto *Direct3D Modo Retenido* y gestionamos la interfaz *IDirect3DRM.*
4. Usamos la función de *Windows GetClientRect* para obtener el alto y ancho de la ventana de la aplicación.
5. Utilizamos el método *CreateDeviceFromClipper*, pasando el valor *NULL* para crear un device por defecto, y pasando el alto y ancho de la ventana de la aplicación al método.
6. Creamos el *frame* principal de la escena utilizando *CreateFrame.*
7. Creamos el *frame* de cámara utilizando *CreateFrame.*
8. Creamos el *Viewport* utilizando *CreateViewport.*
9. Utilizamos la función *BuildScene* para crear la escena.

CÓDIGO FUENTE DE CREATEOBJECTS

```
// Inicializamos globales, crea-
        mos el Device y los objetos.
BOOL CreateObjects(HWND win) {
HRESULT rval;
// Valor de retorno
RECT rc;
// Rectángulo de ventana princi-
        pal para definir tamaño
int width;
// ancho device
int height;
// alto device

// Inicializa toda la estructura
        de variables globales a cero
memset(&myglobs, 0,
        sizeof(myglobs));

// Crea un objeto DirectDrawClip-
        per y le asocia la ventana.
```

```
rval = DirectDrawCreateClip-
    per(0, &lpDDClipper, NULL);
if (FAILED(rval)) {
    DISPLAYMSG("No se pudo crear
    el objeto DirectDrawClip-
    per");
    return FALSE;
}

rval = lpDDClipper->SetHWnd(0,
    win);
if (FAILED(rval)) {
    DISPLAYMSG("No se asignó el
    handle de ventana para el
    objeto DirectDrawClipper");
    return FALSE;
}

// Crea el objeto Direct3DRM.
LPDIRECT3DRM pD3DRMTemp;
rval = Direct3DRMCreate(&pD3DRM-
    Temp);
if (FAILED(rval)) {
    DISPLAYMSG("Falló la crea-
    ción de Direct3DRM.");
    return FALSE;
}
if (FAILED(pD3DRMTemp->
QueryInterface(IID_IDirect3DRM3,
        (void **)&lpD3DRM))) {
    RELEASE(pD3DRMTemp);
    DISPLAYMSG("Error al crear
    Direct3DRM3.\n" );
    return FALSE;
}
RELEASE(pD3DRMTemp);

// Crea un Device por defecto.
GetClientRect(win, &rc);

rval = lpD3DRM->CreateDevice-
    FromClipper(lpDDClipper,
    NULL, // Default Device
    rc.right, rc.bottom,
    &myglobs.dev);
if (FAILED(rval)) {
    DISPLAYMSG("No se pudo crear
    el Device.");
    return FALSE;
}

// Crea el frame principal de la
    escena y el frame de cámara.
rval = lpD3DRM-
    >CreateFrame(NULL,
    &myglobs.scene);
if (FAILED(rval)) {
    DISPLAYMSG("No se pudo crear
    el frame principal. ");
    return FALSE;
}

rval = lpD3DRM-
    >CreateFrame(myglobs.scene,
    &myglobs.camera);
if (FAILED(rval)) {
    DISPLAYMSG("No se pudo crear
    el frame de cámara. ");
    return FALSE;
}

// Creamos el Viewport usando el
    Device, frame de cámara,
// y el ancho y alto del Device.
width = myglobs.dev->GetWidth();
height = myglobs.dev->Get-
    Height();

rval = lpD3DRM->CreateViewport
    (myglobs.dev, myglobs.
    camera, 0, 0,

    width, height,
    &myglobs.view);
if (FAILED(rval)) {
    myglobs.bInitialized =
    FALSE;
    RELEASE(myglobs.dev);
    return FALSE;
}

// Agregamos la ruta para que
    las texturas y objetos .X
    puedan localizarse
AddMediaPath( lpD3DRM );

// Creamos la escena que será
    renderizada.
if (!BuildScene(lpD3DRM,
    myglobs.dev, myglobs.scene,
    myglobs.camera))
    return FALSE;

// Globales son inicializadas.
myglobs.bInitialized = TRUE;

return TRUE;
}
```

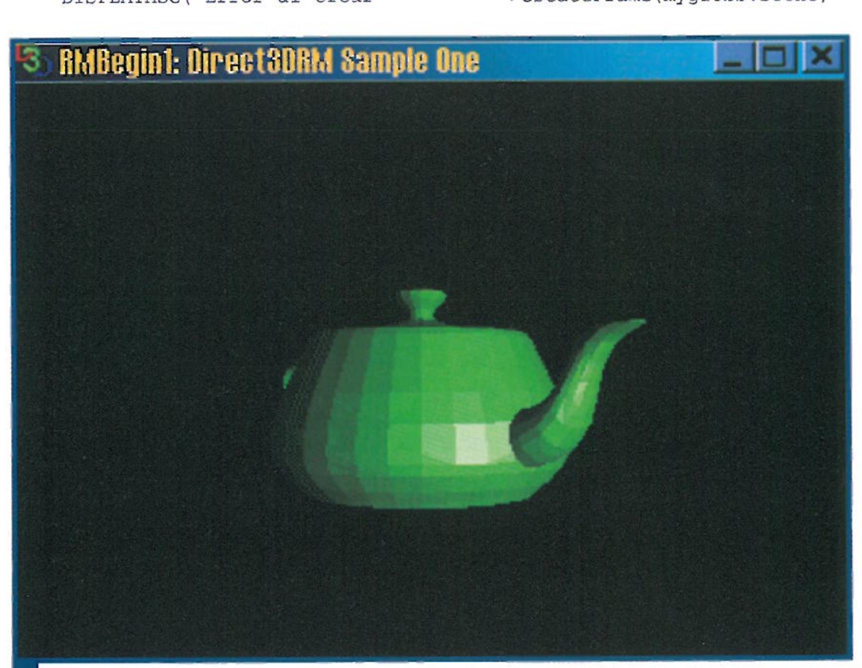


Figura 2: Representación visual del ejemplo tratado en este artículo.

FUNCIÓN INITAPP

Esta función crea la clase de la ventana principal y la ventana correspondiente, tal y como es una aplicación típica de Windows.

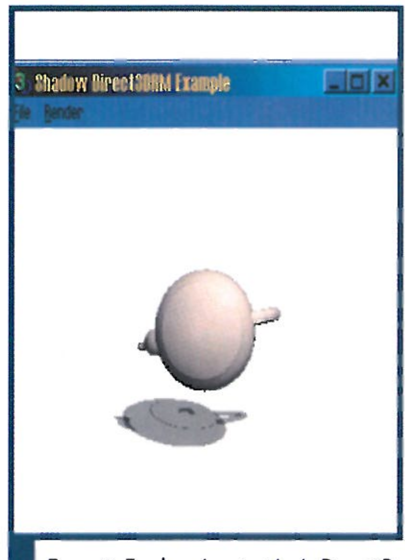


Figura 3: En el modo retenido de Direct3D podemos crear efectos visuales como sombras en tiempo real

CÓDIGO FUENTE DE INITAPP

El código de esta función lo hemos reducido ya que es típico de programas Windows en C++, aunque resaltaremos las partes más importantes del mismo. En el CD-ROM que acompaña la revista podéis encontrar el código completo.

```
// Creamos la ventana principal
     e inicializamos objetos.
static HWND InitApp(HINSTANCE
     this_inst, int cmdshow) {
HWND win;
// Handle de la ventana
WNDCLASS wc;
// Clase de la ventana

// Inicializamos y registramos
     la clase de la ventana.
wc.style = CS_HREDRAW | CS_VRE-
     DRAW;
wc.lpfnWndProc = WindowProc;
...

// Creamos la ventana.
win = CreateWindow(
"D3DRM Example",
// clase
     "RMBegin1: Direct3DRM Sample
     One",
// Nombre
        WS_OVERLAPPEDWINDOW,
    // estilo
        CW_USEDEFAULT,
    // x posición inicial
        CW_USEDEFAULT,
    // y posición inicial
        350,
    // x tamaño inicial
        300,
    // x tamaño inicial
        NULL,
    // Ventana dependiente
        NULL,
    // menu handle
        this_inst,
    // programa handle
        NULL);
    // creación de parámetros

if (!win)
   return FALSE;

// Inicializa variables globales
    y crea objetos de Direct3D
    Modo Retenido
if (!CreateObjects(win))
   return FALSE;
...
}
```

FUNCIÓN WINMAIN

WinMain contiene el bucle de control de la aplicación. Utiliza la función *RenderScene* para dibujar la escena en la pantalla o ventana.

CÓDIGO FUENTE DE WINMAIN

Al igual que en el ejemplo anterior, se ha resumido el código resaltando los aspectos importantes.

```
// Inicializa la aplicación,
    procesa mensajes y renderiza
    la escena.
int APIENTRY WinMain (HINSTANCE
    this_inst, HINSTANCE
        prev_inst,
LPSTR cmdline, int cmdshow) {
 ...

    // Si la aplicación no es
     minimizada y los objetos de
     Direct3D
    // se han inicializado, ren-
     derizamos.

    if (!myglobs.bMinimized &&
     myglobs.bInitialized) {

    // Renderiza un frame de la
     escena. Si éste falla, ini-
     ciamos el mensaje
    // WM_DESTROY, permitiendo a
     la aplicación liberar recur-
     sos antes de terminar la
    // aplicación.

    if (!RenderScene()){
DISPLAYMSG("Fallo en el renderi-
     zado. Abortando ejecu-
     ción.");
PostMessage(NULL, WM_DESTROY, 0,
     0);
break;
    }

...
}
```

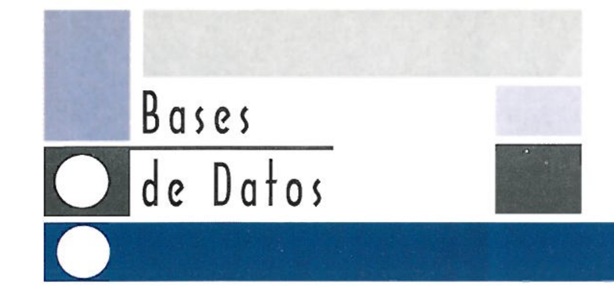
En este artículo hemos aprendido los primeros pasos para poder crear aplicaciones *3D* con *Direct3D*. Con esta base, y un estudio algo más profundo de las posibilidades que nos ofrecen las funciones que componen *Direct3D*, podremos crear sorprendentes programas que hagan uso de las nuevas tecnologías *3D*.

REFERENCIA DE LA SERIE

DIRECTX 6.1

Sólo Programadores 56
INTRODUCCIÓN Y PRINCIPALES NOVEDADES

Sólo Programadores 57
LA UTILIZACIÓN DE DIRECTSOUND



Desarrollo cliente/servidor (II)

Javier Toledo (jtoledo@sei.es)

En esta entrega abordaremos el desarrollo del cliente. Para ello utilizamos la herramienta Visual Basic 6.0. y como base de datos Access.

A partir de este mes comenzamos con un ejemplo de desarrollo de un sistema cliente/servidor en dos capas. Esta entrega se ocupa del desarrollo del cliente. La herramienta elegida ha sido *Visual Basic* 6.0 y la base de datos utilizada *Access*.

LA APLICACIÓN

El ejemplo que a continuación presentamos permite desarrollar nuestra aplicación. Para ello se han omitido problemas que podrían plantearse en un caso real debido a la lógica de nuestro negocio, pero contempla todos los aspectos de una aplicación cliente/servidor.

La integridad de nuestra base de datos será la que se ocupe de que sea imposible llegar a un estado de inconsistencia de datos

Supongamos que somos los propietarios de un conjunto de comercios dedicados a la venta de productos *software*. Nuestros distribuidores y clientes realizan pedidos a cada uno de estos comercios y hemos decidido desarrollar una aplicación que les permita realizar estos pedidos automáticamente. Para ello necesitamos un servidor de bases de datos en el cual figuren cuáles son los productos que tenemos disponibles en cada uno de nuestros comercios y qué cantidad tenemos de los mismos. Partimos del supuesto que nuestros distribuidores tienen acceso a nuestra red de ordenadores. De este modo hemos de almacenar, tanto los distintos pedidos de los clientes, como el stock de nuestros productos.

La integridad de nuestra base de datos será la que se ocupe de que sea imposible llegar a un estado de inconsistencia de datos. Por ejemplo, que en un instante de tiempo tengamos un pedido de uno de nuestros clientes, en el que se nos solicite un producto con el que nosotros habíamos trabajado antes pero ya no lo hacemos.

LOS DATOS

Los datos los hemos almacenado en varias tablas.

- Tabla PRODUCTOS. Almacena los distintos productos.
- Tabla COMERCIOS. Almacena la lista de los distintos comercios y los datos de los mismos.
- Tabla COMERCIOS/PRODUCTOS. Almacena la información de qué productos se encuentran disponibles en un determinado comercio y la cantidad de cada uno de ellos.

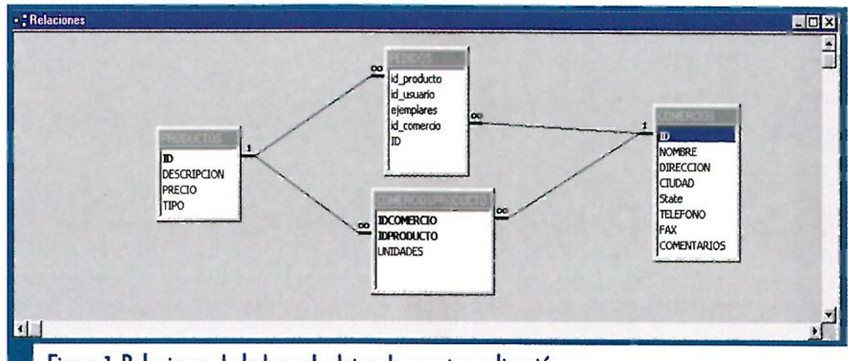


Figura 1. Relaciones de la base de datos de nuestra aplicación.

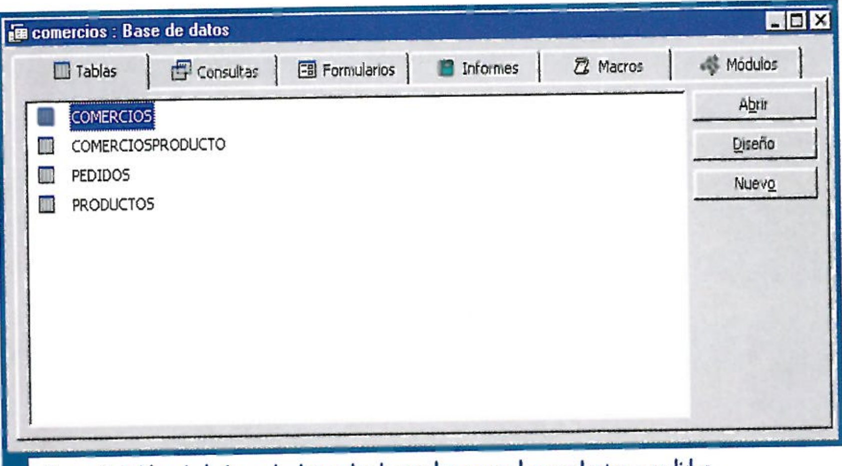


Figura 2. Tablas de la base de datos donde se almacenan los productos y pedidos.

- Tabla PEDIDOS. Almacena los distintos pedidos de nuestros distribuidores y clientes.

PANTALLA PRINCIPAL

Para nuestra pantalla principal hemos decidido presentar nuestros comercios en forma de árbol y mostrar la lista de productos con los que trabajan en una lista a la derecha. Para ello podemos utilizar los controles *Treeview* y *ListView* que vienen en la librería de objetos de *Visual Basic*. También hemos de utilizar dos controles del tipo *Imagelist* que contendrán los gráficos que vamos a mostrar dentro del árbol y la lista respectivamente.

Lo primero que hemos de hacer al arrancar nuestra aplicación es abrir una conexión con nuestra base de datos y obtener la lista de los distintos comercios. Entre las distintas posibilidades de tecnologías de acceso a bases de datos que teníamos para elegir hemos escogido *ADO*, que facilita un modelo de objetos sencillo con un interfaz única.

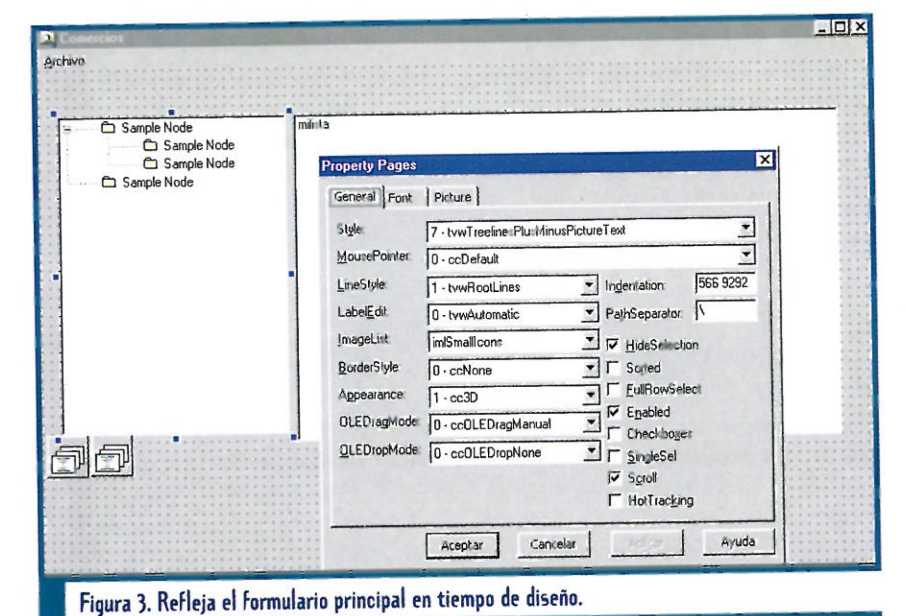


Figura 3. Refleja el formulario principal en tiempo de diseño.

FASES EN EL ACCESO A UN SGBD

Los pasos que ha de seguir cualquier aplicación de *Visual Basic* para tener acceso a un origen de datos utilizando *ADO* son los siguientes:

- Crear un objeto Conexión (*Connection*). Mediante éste vamos a determinar cuál es la base de datos a la que queremos conectarnos y la identificación del usuario de conexión. Un objeto *Connection* representa una sesión con un origen de datos. En nuestro caso sólo utilizamos la base de datos COMERCIOS que ha de encontrarse en el directorioc: *\solop\ejemploc-s\comercios.mdb.*
- Abrir la conexión. Abrimos la conexión con la base de datos.
- Ejecutar sentencias *SQL*. Una vez que hemos abierto la conexión ya podemos consultar y modificar los datos mediante consultas *SQL*.
- Utilizar el conjunto de resultados Tras realizar una consulta *SQL* tenemos disponibles un conjunto de resultados. En función del tipo de cursor, podemos examinar y modificar los datos en el servidor o en el cliente.
- Finalizar la conexión Cerramos nuestra conexión con el servidor de datos.

CREAR LA CONEXIÓN

Lo primero que hemos de hacer cuando arranque nuestra aplicación es crear una conexión con el *SGBD* que pensemos utilizar. Para ello insertaremos el código encargado de crear una nueva instancia del objeto *Connection* en el evento

Form_Load de nuestro formulario principal, y establecemos la propiedad *ConnectionString* para especificar un origen de datos. Una vez creado el objeto, ejecutaremos el método *open* para abrir la conexión.

```
Private Sub Form_Load()
     sDirectorio =
     "c:\solop\ejemploc-s\"
     inUsuario = 1

     ' Abrir la conexión.
     On Error GoTo HayError

     Set cn = New ADODB.
     Connection
     ' ConnectionString contiene
     la ruta de la base de datos.
     ' El archivo comercios.mdb
     debe estar en el directorio
     c:\solop\ejemploc-s
     With cn
          .ConnectionString =
     "Provider=Microsoft.Jet.
     OLEDB.3.51;
Data Source=" & _
     sDirectorio &
     "comercios.mdb"
          .Open
     End With

     ' Configurar la lista.
     milista.View = lvwIcon

     ' Configurar el árbol
     With arbol
     .Sorted = True
     Set mNode = .Nodes.Add()
     .LabelEdit = False
     .LineStyle = tvwTreeLines
     End With

     With mNode ' Agregar el nodo
     raíz.
     .Text = "comercios"
     .Tag = "LISTADOcomercios"
     .Image = "closed"
     End With
     Show
     Cargar_arbol ' ejecuta la
     función que genera el árbol
     Exit Sub

  HayError:
   MsgBox "Ha ocurrido un error.
   Compruebe que comercios.mdb
   se encuentra en el directorio
   c:\solop\ejemploc-s\ "

 Unload frmPrincipal

 End Sub
```

El siguiente paso consiste en insertar un nodo en nuestro árbol por cada comercio. Para ello hemos de obtener la lista de los comercios utilizando la siguiente instrucción SQL:

```
SELECT ID, DIRECCION FROM COMERCIOS
```

Necesitamos crear una instancia del objeto *Recordset*, el cual nos va a permitir manejar el conjunto de datos resultantes de nuestra consulta. Podemos abrir un objeto *Recordset* (es decir, ejecutar una consulta) sin tener que abrir de forma explícita un objeto *Connection*. Pero creando previamente la instancia del objeto Connection podremos abrir múltiples objetos *Recordset* en la misma conexión.

```
 ' Crear un objeto Recordset de
   ADODB.
    Dim rscomercios As New
   ADODB.Recordset

 ' Abrir el Recordset seleccio-
   nando las distintas direccio-
   nes de nuestros comercios.
   With rscomercios
      .Open "SELECT ID, DIREC-
   CION FROM comercios",
   cn, adOpenStatic,
   adLockOptimistic
    End With
```

Una vez ejecutada la consulta SQL tenemos que ir desplazándonos por las filas resultantes de la consulta, y por cada una de ellas generar un nuevo nodo en el árbol. Para desplazarnos entre el conjunto de las filas resultantes podemos utilizar los métodos *MoveFirst*, *MoveLast*, *MoveNext* y *MovePrevious*. Podemos detectar cuando hemos llegado al final de un *Recordset* chequeando la propiedad *Recordset.EOF*.

Una vez que tenemos inicializado el *Recordset* y nos hemos situado en la fila que nos interesa, podríamos seleccionar un campo de esa fila mediante *Recorset!ID* y *Recordset!DIRECCION*.

```
Private Sub Cargar_arbol()
    ' Crear una variable Record-
    set de ADODB.
    Dim rscomercios As New
    ADODB.Recordset

     'Abrir el Recordset selec-
    cionando las distintas direc-
    ciones de los comercios.

    With rscomercios
         .Open "SELECT ID,
    DIRECCION FROM comercios",
    cn, adOpenStatic,
    adLockOptimistic
    End With

     ' Insertar un nodo en el
    árbol para cada comercio
     Do While Not rscomercios.EOF

     ' Agregar un nodo al árbol y
    asignarle "comercios" como
    tag.
     Set mNode =
    arbol.Nodes.Add(1, tvwChild,
    rscomercios!ID & " ID",
CStr(rscomercios![DIRECCION]),
    "closed")
     mNode.Tag = "comercios"
     rscomercios.MoveNext
    ' Cambiamos al siguiente
    comercio.

    Loop

    rscomercios.Close

     ' Ordenar los nodos.
     arbol.Nodes(1).Sorted = True
     ' expandimos el arbol.
     arbol.Nodes(1).Expanded =
    True

End Sub
```

Una vez que tenemos el árbol cargado con los comercios hemos de capturar los eventos *Collapse* y *Expand* y escribir el código necesario para que el icono del nodo raíz cambie de una carpeta abierta a una carpeta cerrada. Esto podemos hacerlo cambiando la propiedad *Image* del nodo raíz. Para que el programa localice las imágenes closed y open hemos de haber insertado estas imágenes en un control *ImageList* en tiempo de diseño. También debemos asociar dicho *ImageList* en tiempo de diseño a nuestro *TreeView* introduciendo el nombre del *ImageList* en la propiedad con el mismo nombre que se despliega pulsando con el botón derecho sobre el *Treeview* y pulsando *Properties* en tiempo de diseño.

```
Private Sub arbol_Collapse(ByVal
    node As node)
     ' Cambiar el icono de carpe-
     ta.
      node.Image = "closed"
End Sub

Private Sub arbol_Expand(ByVal
    node As node)
' Cambiar el icono de carpeta.
    node.Image = "open"
End Sub
```

Cuando el usuario seleccione un comercio hemos de cargar nuestro *ListView* con la lista de productos existentes en ese comercio. Para obtener la relación de los distintos productos utilizaremos la instrucción SQL:

```
SELECT COMERCIOS.DIRECCION,
    COMERCIOS.CIUDAD,
    COMERCIOS.TELEFONO,
    COMERCIOS.FAX, PRODUCTOS.ID,
    PRODUCTOS.DESCRIPCION,
    COMERCIOSPRODUCTO.UNIDADES
    FROM
       (COMERCIOSPRODUCTO INNER
    JOIN PRODUCTOS ON
              COMERCIOSPRODUC-
    TO.IDPRODUCTO =
    PRODUCTOS.ID)
        INNER JOIN COMERCIOS ON
    COMERCIOSPRODUCTO.IDCOMERCIO
    = COMERCIOS.ID
 WHERE COMERCIOS.ID = XXXX.
```

Donde XXXX es el identificador del comercio que el usuario ha seleccionado. La función *MakeColumns* (el código está disponible en las fuentes de la revista) se encarga de preparar el control *ListView* para que almacene datos complementarios en cada item de la lista. Por ejemplo, el identificador del producto y del comercio.

La función *Contenido_comercios* es invocada desde la función que atiende al evento *NodeClick*. Pasándole como argumento el identificador del producto seleccionado. El método *AddListItem* es el encargado de insertar en el *ListView* los productos.

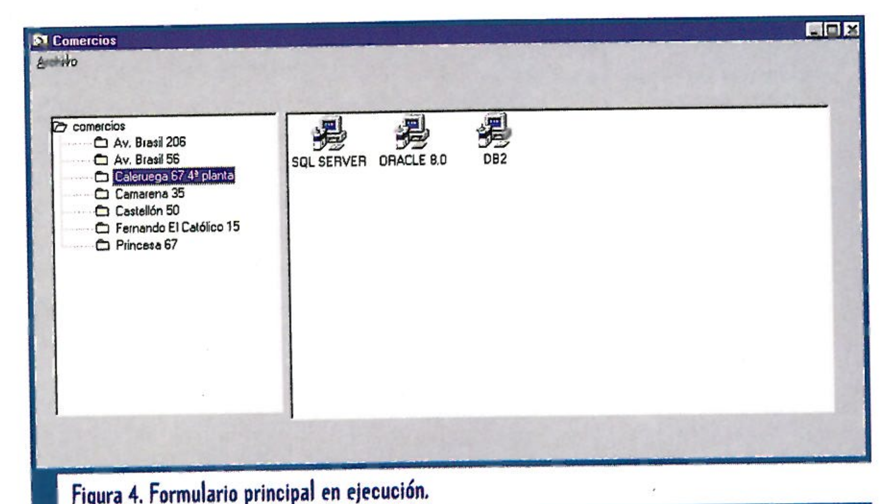


Figura 4. Formulario principal en ejecución.

```
Private Function
    Contenido_comercios(ID As
    Integer) As Boolean
     Dim rsProductos As New
    ADODB.Recordset
     Dim newNode As node
     ' Para nuevo nodo.
     Dim strQ As String

     ' Si ListView ya está lleno,
    lo borramos.
     milista.ListItems.Clear

     ' Esta función es la encar-
    gada de crear las columnas
donde almacenaremos datos
  ' complementarios de un pro-
ducto para poder utilizarlos
más adelante.

 MakeColumns

 ' Consulta SQL que obtiene
todos los campos necesarios.
 strQ = "SELECT
    COMERCIOS.DIRECCION,
COMERCIOS.CIUDAD,COMERCIOS.
TELEFONO,

COMERCIOS.FAX,PRODUCTOS.ID,
PRODUCTOS.DESCRIPCION,
    COMERCIOSPRODUCTO.UNIDA-
DES FROM (COMERCIOSPRODUCTO
INNER JOIN PRODUCTOS ON
COMERCIOSPRODUCTO.IDPRODUCTO
= PRODUCTOS.ID)
        INNER JOIN COMERCIOS
ON COMERCIOSPRODUCTO.IDcomer-
cio = COMERCIOS.ID
     where COMERCIOS.ID=" & ID

  ' Abrimos el Recordset.
Salimos si no hay resultados.
 With rsProductos
      .Open strQ, cn, adO-
penStatic, adLockReadOnly,
adCmdText
      If .BOF Then
          ' Si no hay resul-
tados, devuelve False y sale
Contenido_comercios = False
          Exit Function
```

```
        End If
    End With

    On Error GoTo ComerciosErr

    ' Agregar el objeto ListItem
correspondiente.
    AddListItem mItem, rsProduc-
tos, ID

    rsProductos.MoveNext

    ' Recorremos el Recordset.
agregando un objeto ListItem.
    Do Until rsProductos.EOF
        AddListItem mItem,
rsProductos, ID
        rsProductos.MoveNext
    Loop

    Contenido_comercios = True '
devuelve True si funciona
correctamente
    Exit Function

ComerciosErr:
        Debug.Print Err.Number,
    Err.Description
        Debug.Print rsProduc-
    tos!ID
        Resume Next

    Exit Function
End Function
```

FORMULARIO DE PEDIDO

Una vez que el usuario ha seleccionado un comercio y un producto tenemos que permitirle realizar pedidos. Los pedidos de un usuario se almacenan en la tabla *PEDIDOS*, en la que cada fila representa el pedido de un cliente. Los campos de cada registro son los siguientes: Identificador del pedido, Identificador del usuario, Identificador del comercio al que se le hace el pedido, Identificador del producto que se solicita y el número de unidades solicitadas.

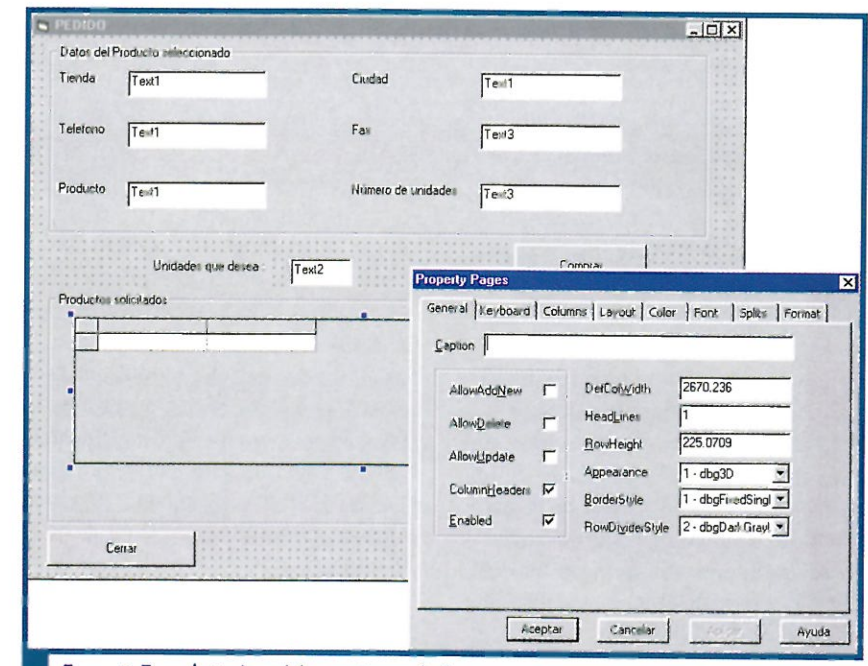


Figura 5. Formulario de pedidos en tiempo de diseño.

TRANSACCIONES

Cuando un usuario realice un pedido tendremos que efectuar dos pasos:

- Insertar un nuevo registro en la tabla pedidos.
- Disminuir el número de productos disponibles en el comercio al cual ha solicitado el producto.

Al realizar todas las transacciones es importante la seguridad ante posibles fallos

Con este tipo de operaciones es importante que establezcamos algún mecanismo de seguridad ante posibles fallos. ¿Qué ocurriría si el programa falla cuando hemos insertado el pedido de un cliente pero no hemos disminuido el número de productos disponibles en el comercio? Podría darse el caso de que cuando fuésemos a mandar a este cliente su pedido no tuviésemos suficientes productos. Para evitar esto existen las transacciones.

Las transacciones constituyen un mecanismo que permiten a la base de datos crear bloques de elementos que se ejecuten como un único grupo. Con que falle una transacción falla todo el grupo y se deshacen todas las acciones realizadas dentro de esta transacción. El resultado final será como si ninguna de las operaciones de la transacción se hubiera realizado.

Para efectuar transacciones con ADO tenemos que disponer de un objeto *Connection*. Los métodos de este objeto para administrar una transacción son los siguientes:

- *BeginTrans* inicia una nueva transacción.
- *CommitTrans* guarda las modificaciones efectuadas y termina la transacción actual.
- *RollbackTrans* cancela las modificaciones efectuadas durante la transacción actual y termina la transacción.

Dependiendo de la propiedad *Attributes* del objeto *Connection* que utilicemos, la llamada a los métodos *CommitTrans* o *RollbackTrans* puede iniciar automáticamente una nueva transacción.

MÉTODOS PARA EJECUTAR COMANDOS CONTRA UNA BASE DE DATOS

Cuando queremos modificar registros de una base de datos podemos hacerlo de dos modos, obteniendo un *Recordset* con los registros que nos interesan y modificando el contenido del mismo o mandando directamente una instrucción SQL que modifique o añada registros. Los métodos son los siguientes:

- *Connection.Execute*, el comando es una cadena *SQL*. En nuestro ejemplo lo utilizamos tanto para realizar operaciones de inserción *(insert)* como de actualización *(updates)* en la tabla **PEDIDOS**.
- *Command.Execute*, antes de ejecutar este método hemos de establecer la propiedad *Command.CommandText* como la instrucción *SQL* que queremos utilizar. Tanto este método como el anterior están previstos para instrucciones que no devuelven datos, es decir instrucciones de modificación, no de consulta. (Aunque pueden utilizarse para ambas posibilidades). Devuelven objetos *Recordset* del tipo *Fordward-only* que son de cursor estático.
- *Recordset.Open*, el comando es el argumento Source, que puede ser una instrucción *SQL* o un objeto *Command*. Permite especificar el *CursorType* (estrategia de acceso a los datos) y el *LockType* (el tipo de bloqueo y de actualizaciones en los modos inmediato o por lotes).

BLOQUEOS Y EL ACCESO MULTIUSUARIO

En una base de datos multiusuario nos encontramos con el problema del acceso concurrente a la información. Los distintos usuarios pueden intentar modificar el mismo registro. ¿Cuál es el resultado final de la base de datos si un usuario está modificando el precio de un producto mientras otro está eliminando dicho producto? ¿deberíamos permitir que dos usuarios estén modificando el mismo registro simultáneamente? La herramienta que hemos de utilizar en estos casos es el bloqueo.

El concepto de bloqueo es el siguiente: en el momento en que un usuario edita un registro se convierte en el propietario del mismo y el resto de los usuarios no pueden modificarlo. En el momento en el que el usuario termina la edición sobre el registro y libera la propiedad de bloqueo otro usuario puede apoderarse igualmente del registro. Evidentemente el que un registro esté bloqueado no impide al resto de los usuarios lanzar consultas sobre este registro, pero sí les impide los intentos de modificación del mismo.

BLOQUEOS OPTIMISTAS Y PESIMISTAS

El bloqueo pesimista se establece cuando se comienza a editar los datos. En ese momento se bloquean los registros necesarios de la base de datos y no serán liberados hasta que llamemos a la instrucción *Update*. En cambio el bloqueo optimista se realiza únicamente durante el tiempo que tarda en ejecutarse el *Update*. La propiedad *LockType* se ocupa de definir cuál es el tipo de bloqueo con el que trabajará el *Recordset*. Dicha propiedad puede tener los siguientes valores:

- *adLockReadOnly*. Los datos del Recordset son de sólo lectura.
- *adLockPessimistic*. Bloqueo pesimista.
- *adLockOptimistic*. Bloqueo optimista.
- *AdLockBatchOptimistic*. Actualiza varios registros simultáneamente con el método *UpdateBatch*.

En nuestro ejemplo hemos realizado las operaciones de actualización utilizando directamente instrucciones *SQL* (sin *Recordsets*) por lo tanto el tipo de bloqueo utilizado ha sido el optimista. Cada usuario de nuestra aplicación debe ser identificado mediante un identificador de tipo *Integer*. En nuestro ejemplo establecemos por defecto el valor **1** a la variable *inUsuario*, variable que almacena dicho identificador.

Una base de datos del tipo de *Access* realiza el bloqueo por páginas, mientras que *Oracle* permite realizar dichos bloqueos a nivel de registro. El bloqueo por páginas no es un bloqueo de registro puro.

CONCLUSIÓN

Hasta aquí hemos visto el desarrollo del cliente. En la próxima entrega de la serie realizaremos el proceso de diseño y creación de la base de datos, tanto en *SQL Server* como en *Oracle*.

REFERENCIA DE LA SERIE

Desarrollo cliente/servidor (I)

Sólo Programadores 57

DEFINICIÓN, COMPONENTES, MÉTODOS Y TECNOLOGÍAS

VisualAge C++ 4.0. La apuesta de IBM en C++

Javier Sanz Alamillo (jsanza@teleline.es)

La versión 4 de este producto ofrece todo un conjunto de revolucionarias características y nuevos conceptos para mejorar el desarrollo en C++.

INTRODUCCIÓN

Si bien *VisualAge C++* no es un producto que se encuentre entre los entornos más conocidos para desarrollar en *C++*, esta última versión puede hacer que más de uno considere su utilización antes que otro más renombrado, que no por ello mejor. *IBM* ofrece mediante *VisualAge C++* un entorno *IDE* de desarrollo en *C/C++* con un gran conjunto de asistentes y herramientas, todo ello enfocado al desarrollo de aplicaciones multiplaforma orientadas a objeto. El principal objetivo es que se disponga de un entorno que permita mejorar tanto la calidad como reducir el tiempo de creación en los proyectos.

CARACTERÍSTICAS PRINCIPALES

Las características principales de este nuevo entorno de desarrollo son las siguientes:

- Un entorno *RAD* que integra un editor, un compilador incremental, un *debugger*, así como asistentes y herramientas visuales que facilitan muchas tareas repetitivas. Muy similar, gráficamente y conceptualmente al utilizado por los programadores *Java*, el conocido *VisualAge for Java 2.0*.
- Un compilador incremental y una herramienta de optimización que mejoran todo el proceso de compilación y "linkado" de forma sorprendente, especialmente a la hora de compilar grandes proyectos, en lo que es la muestra "del paso adelante" en el desarrollo de los compiladores tradicionales.
- Mediante las herramientas *Visual Builder* y *Data Access Builder* se pueden construir entornos gráficos y aplicaciones que acceden a bases de datos de forma rápida y sencilla.
- Un conjunto de clases denominadas *IBM Open Class Library*, diseñadas para la creación de aplicaciones multiplataforma entre *AIX*, *OS/2* y *Windows*.
- Un sistema de ayuda al programador basado en *HTML*.
- Y otra importante novedad, es que cumple el nuevo estándar 1998 *ANSI/ISO C++*, incluso a nivel de librería de clases.

Aquellos proyectos que están utilizando alguna versión anterior de *VisualAge* o de un compilador *C/C++* de *IBM* y vayan a comenzar un nuevo proyecto sin duda deberían utilizar este nuevo entorno .El producto incluye un nuevo compilador *C/C++* versión *3.6*, disponible para *Windows NT* y OS/2.

VisualAge presenta un interesante conjunto de mejoras

Además, se puede disponer del programa *GreatCircle*, un *code-checker*, que viene incluido en el producto (aunque únicamente en versión de evaluación) y que permite al programador corregir problemas en el código, como *memory leaks*, anomalías relacionados con la fragmentación de la memoria, control sobre problemas con *software* de terceras partes, etc.

INSTALACIÓN

La instalación de *VisualAge* C++ es una tarea que se basa en el seguimiento de las instrucciones que va mostrando el asistente disponible para ello, y tras seleccionar unas cuantas opciones, como el directorio de destino, el tipo de instalación, etc., se dispone de este nuevo entorno para comenzar a trabajar. Mencionar que hay disponible una opción para la instalación de *VisualAge* a través de red, la cual no es muy recomendable porque hace que este proceso sea algo lento y que existe la posibilidad de utilizar un modo de instalación sin preguntas, denominado *Silent Installation,* mediante el cual el proceso de instalación se automatiza, aún más si cabe, al utilizar un fichero con las respuestas a todas las opciones que se presentan al usuario.

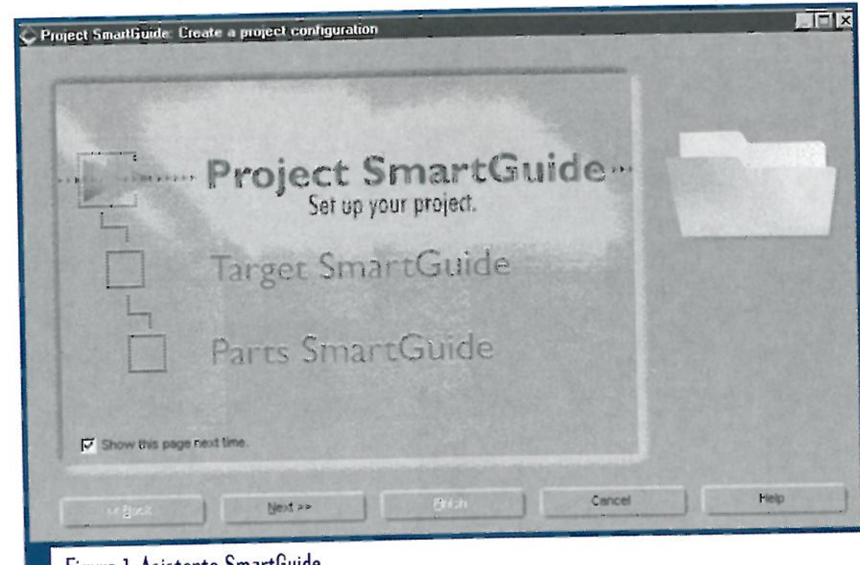


Figura 1. Asistente SmartGuide.

REQUERIMIENTOS

A la hora de hablar de los requerimientos necesarios para la instalación y utilización es cuando los desarrolladores quedarán algo sorprendidos. A nivel de *software* no se exigen requisitos especiales o fuera de lo común. Disponer de un sistema operativo *Windows NT 4.0* (no se especifica ninguna versión de *Service Pack*) o *OS/2 Warp* versión *4.0*, tener instalado un navegador de *Internet*, como por ejemplo, *Netscape Navigator* para visualizar la ayuda disponible en *HTML* y si se va a utilizar bases de datos, tener un gestor como *DB2* u otro cualquiera que soporte *ODBC* instalado.

Hasta este momento nada fuera de lo esperado, pero es el *hardware* requerido lo que sorprende. El *hardware* requerido para utilizar *VisualAge* cómodamente es algo superior a lo esperado y posiblemente superior al actualmente disponible para la mayoría de los desarrolladores. Básicamente, los requerimientos hardware necesarios o que se deberían disponer para el correcto funcionamiento son los siguientes:

- El procesador mínimo requerido es un *Pentium* a *166 Mhz* o superior. Tras varias pruebas, el uso de un P166 se "queda" algo justo, por lo que no es muy recomendable nada menor a 200 *Mhz*, es más, un *Pentium II* a 300 *Mhz* sería el procesador ideal para trabajar cómoda y velozmente.
- La *RAM* que se requiere es de 64 *Mb* mínimo, y se recomiendan 128 *Mb*. En este punto se debería seguir la recomendación fielmente, puesto que con el uso de 64 *Mb* todas las tareas parece que se ralentizan, no se encuentra cierta soltura como se espera.
- De vídeo se recomienda una *SVGA*, de resolución mínima *800x600*, aunque se especifica que lo ideal sería *1024x768*. Claramente se aprecia a los pocos segundos de la instalación que este producto se ha diseñado para la segunda resolución, por lo que el uso de *800x600* hace que las áreas de texto, menús, etc., aparezcan a veces con escasas opciones y hay que usar demasiado el ratón para buscar las cosas.

Como se observa, los requerimientos *hardware* son algo elevados, aunque en cierta medida, el resultado en comparación con los que puedan ofrecer otros entornos de desarrollo compensa estas "exigencias". Del tamaño de disco necesario para su instalación, siguiendo la tónica general de todos los productos similares que a veces parece que sea ocupar todo lo disponible, tenemos que el espacio libre requerido es:

- En el caso de *Windows NT*, la instalación completa (y recomendada) de *VisualAge* C++ *4.0* ronda los 540 *Mb*. En el caso de instalar el compilador *C/C++* versión *3.6* el tamaño requerido es de 500 *Mb*.
- En el caso de *OS/2*, *VisualAge* ocupa unos 590 *Mb* y para la instalación del compilador *C/C++* aproximadamente unos 460 *Mb*.

Se puede decir que los requerimientos *hardware* para este entorno son bastante elevados, aunque si se mira desde otro punto de vista, también lo son las ventajas y comodidades que ofrece.

CARACTERÍSTICAS GENERALES

VisualAge C++ presenta todo un amplio conjunto de mejoras y nuevas posibilidades que hacen que el

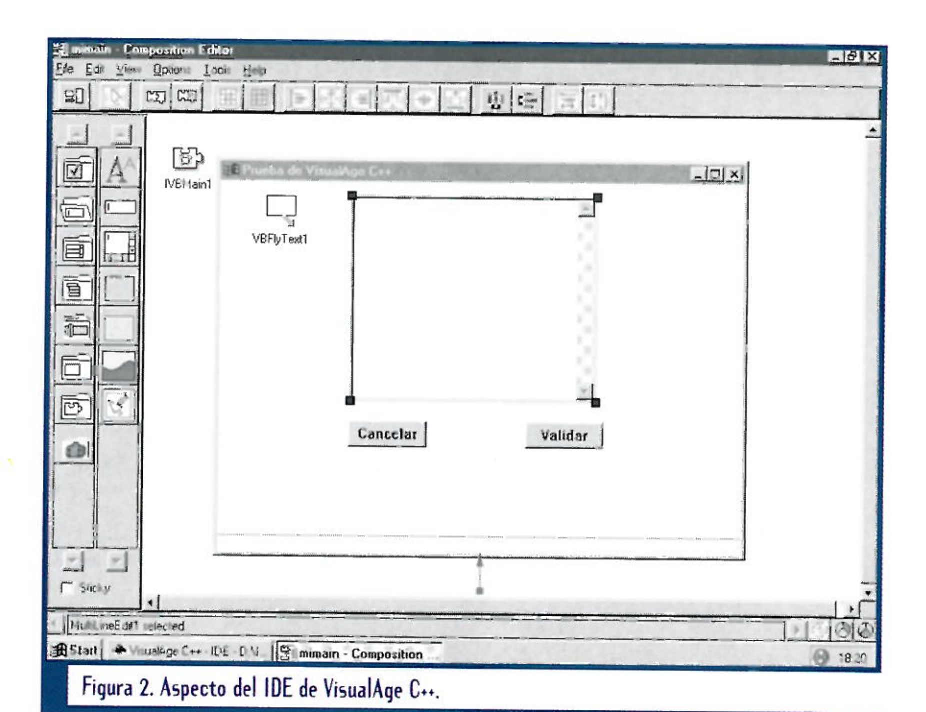


Figura 2. Aspecto del IDE de VisualAge C++.

entorno de desarrollo sea cómodo, práctico, de fácil aprendizaje (sobre todo si se ha trabajado con productos *IBM*, ya que sigue una cierta filosofía corporativa, como con el producto *VisualAge for Java*) y potente para desarrollar aplicaciones de cualquier tipo, visuales, con acceso a bases de datos, con posibilidades de distribución, etc.

Basado en un entorno *RAD* bastante integrado, lo que llama más la atención inicialmente es el ahorro de tiempo en la compilación, gracias al compilador incremental, la posibilidad de no utilizar ficheros de compilación *makefiles*, las nuevas mejoras al desarrollar código, el manejo de las *templates*, la facilidad para corregir errores gracias a unos mensajes de aviso y error acertados y que no despistan al programador. Veamos todo el conjunto de posibilidades y características más detalladamente.

EL ENTORNO RAD DE DESARROLLO

El nuevo entorno presente en esta versión mejora la forma en que se desarrollan los programas. Mediante las diferentes posibilidades, el programador puede manejar mejor los ficheros utilizados de forma independiente, personalizar la forma de trabajo, las opciones que utilizar, etc. Permite visualizar gráficos que muestran las relaciones entre clases y la sobrecarga. Consta de un *debug* cómodo, eficiente y de una serie de asistentes, como el *SmartGuide*, utilizados para desarrollar la mayoría de las tareas eficazmente.

Mediante el asistente *SmartGuide* se simplifica el desarrollo de las acciones más complicadas. A través de una serie de selecciones que gestiona *SmartGuide* de acuerdo a unas especificaciones se consigue gestionar proyectos (*Project SmartGuide*), relacionar ficheros y datos, (*Target SmartGuide*), y crear aplicaciones visuales, mediante el uso de la herramienta *Visual Builder*, que accedan a bases de datos, basándose en otra herramienta, *Data Access Builder*, lo que conforma el denominado *Parts SmartGuide*.

COMPILACIÓN INCREMENTAL

Una de las nuevas características es la velocidad con la que compila y construye los ejecutables. Si bien en las referencias propias del producto indican que en grandes proyectos el compilador puede llegar a ser cerca de diez veces más rápido que los tradicionales, éste rápidamente se da cuenta de que hay algo nuevo que hace que todo marche con más fluidez, las recompilaciones son verdaderamente más rápidas que en otros entornos.

- El tiempo de compilación debería ser proporcional al número de cambios hechos en el código fuente hasta la última compilación. Esto significa que realizar un cambio en los comentarios no requiere recompilar, que una modificación en una función no *inline* no requiere más que recompilar esa función. Un cambio en una función *inline* sólo requiere el cambio en las funciones que la llaman, no en todas las restantes que no tienen relación con ella, etc.
- La cantidad de tiempo utilizada por el *linker* debería ser proporcional al tamaño de las funciones recompiladas, no proporcional al tamaño del programa.

CREACIÓN Y MANTENIMIENTO DE CÓDIGO

En este entorno se puede desarrollar código y mantenerlo mucho más fácilmente que en versiones anteriores, e incluso que en otros entornos similares de desarrollo C++. Mediante *VisualAge* C++ se puede realizar la programación del tipo *Orderless Programming*, es decir, el programador no tiene porqué declarar las funciones antes de ser llamadas o definir las clases antes de usarlas.

También este tipo de programación implica algo que sorprenderá a la mayoría, y es que no hay necesidad de realizar todo el ritual de escribir la lista de *#includes* al inicio de los programas. Veamos un pequeño ejemplo. Supongamos que tenemos los ficheros: **A.h**, **B.h** y **main.cpp**.

```
file A.h
#ifndef A_H
#define A_H
#include "B.h"
class A:public B {
   public:
      B* f() { return new B; }
}
#endif
file B.h
#ifndef B_H
#define B_H
#include "A.h"
class B {
   public:
      A* g() { return new A; }
}
#endif
file main.cpp
#include "B.h"
int main() {
   B b;
   return 0;
}
```

Este código muestra cómo se organizan básicamente los ficheros de cabecera: se define cada clase en un fichero *header* y se utiliza una macro para evitar la múltiple inclusión, añadiendo el *#include* de cada clase necesaria. Pero aquí se presenta un problema.

El simple código anterior no compila cuando se organiza de esta forma. Para remediarlo se tiene que añadir la clásica declaración adelantada, por ejemplo, la clase **A** en el fichero **B.h** y así realizar toda una serie de cambios que suelen ser bastante fastidiosos sobre todo en proyectos con muchas clases y ficheros. Mediante el uso de *VisualAge* no hace falta realizar ningún artificio ni similares. El programador puede organizar los ficheros en cualquier orden, *VisualAge* se encargará del resto. En resumen, se puede realizar una *orderless programming*. Así, el código anterior se podría reescribir de la siguiente forma:

```
file A.h
class A:public B {
   public :
      B* f() { return new B; }
}
file B.h
class B {
   public :
      A* g() { return new A; }
}
file main.cpp
int main () {
   B b;
   return 0:
}
```

Como se observa, en el código no aparecen declaraciones anticipadas ni macros ni *#includes*, eliminándose la desagradable tarea de corregir ciertos errores. Si el programador desea seguir con su forma de trabajo tradicional tampoco encuentra ningún problema de uso.

DESARROLLO MULTIPLATAFORMA CON OPEN CLASS LIBRARY

Mediante el conjunto de clases que ofrece *Open Class Library* el programador puede desarrollar todo un conjunto de aplicaciones portables entre entornos *AIX*, *Windows*, *AS/400* y *OS/2*. Esta librería ofrece posibilidades de internacionalización de código, gestión de distintos sistemas de archivos, diseño de aplicaciones *2D*.

CREACIÓN DE INTERFACES Y APLICACIONES VISUALES

Usando la herramienta *Visual Builder* se pueden construir aplicaciones visuales de una forma sencilla y práctica. Se dispone de un gran número de componentes reutilizables, de un sistema de construcción basado en el sistema evento-interacción, que los programadores *Java* que usen *VisualAge for Java* conocerán perfectamente, así como apreciarán sus ventajas una vez que se familiarizen con su uso, ya que supone un mayor esfuerzo en tiempo y desarrollo.

GESTIÓN DE BASES DE DATOS

Mediante el uso de *Data Access Builder(DAB)* se pueden crear relaciones entre clases *C++* y tablas de forma que *DAB* genere el código necesario para gestionarlas. Este proceso es similar al requerido cuando se utiliza *DBTools++* de *RogueWave*, lo que permite integrar más el programa con la gestión de las bases de datos, sin necesidad de usar *PRO-C* ni similares, con los problemas que a veces ello conlleva. Mediante *DAB* se pueden realizar cómodamente las siguientes tareas:

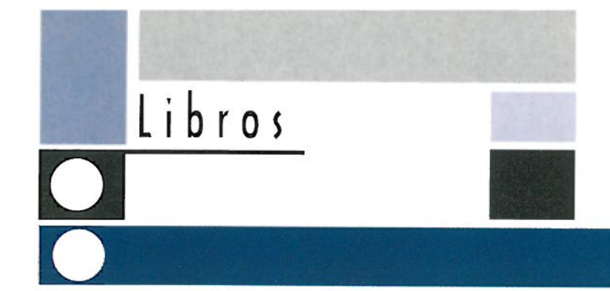
- Separar los procesos de conexión y desconexión a las bases de datos.
- Gestionar operaciones de *Commit* y *Rollback*.
- Seleccionar y recuperar grupos de objetos de la base de datos mediante el uso de *Open Class Library*.
- Mediante *Open Database Connectivity* se puede acceder a múltiples gestores de bases de datos a través de *ODBC* mediante el *driver* adecuado.

SOPORTE DEL ÚLTIMO ESTÁNDAR C++

Se han añadido interesantes características en el nuevo estándar *ANSI C++*, entre las que se encuentran por ejemplo el *RTTI* (*Run-Time Type Identification*), excepciones y cambios en las *templates* y la sobrecarga. *VisualAge C++* soporta todas las nuevas especificaciones definidas en el estándar *ANSI/ISO* del febrero de 1998, excepto por aquellas que se relacionan con la *ordeless programming*.

CONCLUSIÓN

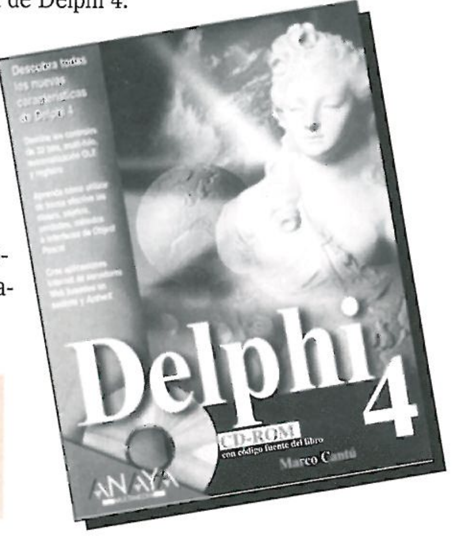
Como se ha podido comprobar, *VisualAge C++ 4* se presenta como un buen producto para el desarrollo en entornos *RAD* con *C++*. Gracias a sus novedades, el desarrollador observará cómo gana mucho tiempo en tareas que anteriormente asumía como lentas. Mediante las herramientas y los asistentes se pueden construir aplicaciones gráficas de forma sencilla, a la vez que realizar aplicaciones que gestionen bases de datos de una forma potente y práctica.



LA BIBLIA DE DELPHI 4

Esta importante editorial nos acerca de nuevo al fascinante mundo del lenguaje Delphi. En esta ocasión nos presentan un título llamado La Biblia de Delphi 4.

Un volumen que abarca desde un nivel medio de programación hasta el más avanzado, todo aquí tiene cabida. Se tratan en profundidad las últimas mejoras que se han introducido en la arquitectura de Delphi 4.



Aprenderás a estudiar y reutilizar cientos de ejemplos prácticos ejecutables compilados para esta versión, también conseguirás utilizar en tus programas cualquiera de los objetos que se construyan a lo largo de los capítulos del libro. Lograrás utilizar de un modo efectivo los controles de 32 bits, la bandeja de iconos y el Registro, las clases, los objetos, las unidades y los métodos.

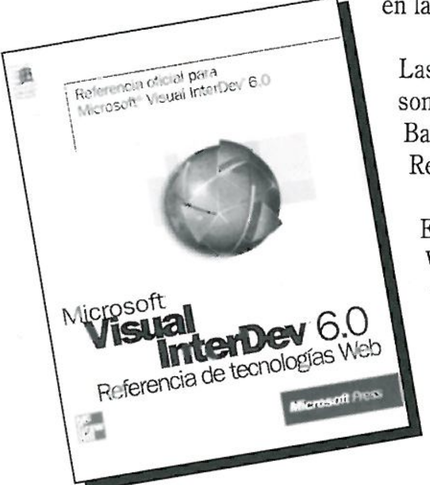
Aprovecharás al máximo las funciones visuales de herencia de formularios, el Depósito de Objetos (Object Repository) y las cualidades de todos los elementos avanzados y novedosos.

Editorial: Anaya Multimedia
Nº páginas: 864
Nivel: Medio-avanzado

Autor: Marco Cantú
Idioma: Español
Precio: 5.995 Ptas. (I.V.A. inc.)

MICROSOFT VISUAL INTERDEV 6.0

Este libro pretende ser una referencia indispensable para construir aplicaciones Web dinámicas. Se trata de un completo conjunto de recursos, recopilado en un único volumen que ofrece un total de cinco guías esenciales sobre los diferentes lenguajes de script aplicables tanto por el lado del servidor como por el lado del cliente.



Este manual te enseñará el entorno de programación de Visual InterDev 6.0 y te guiará en la creación de proyectos Web y en la utilización de bases de datos.

Las cinco guías de las que hablamos y que son las que componen este amplio ejemplar son las detalladas a continuación: Dynamic HTML, Microsoft Jscript, Microsoft Visual Basic Script, Referencia de objetos y controles de Microsoft Visual InterDev y la Referencia ADO.

Este volumen permite a los desarrolladores de todos los niveles crear aplicaciones Web altamente interactivas y orientadas a los datos a través de un entorno completamente visual y totalmente accesible por cualquier explorador o plataforma.

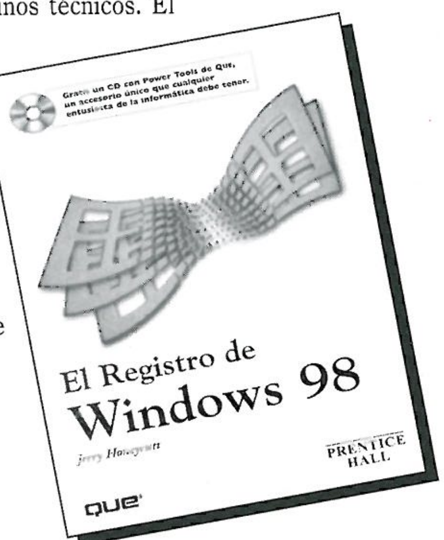
Editorial: McGraw-Hill
Nº páginas: 1604
Nivel: Todos los niveles

Autor: Microsoft Press
Idioma: Español
Precio: 10.500 Ptas.(I.V.A. inc.)

EL REGISTRO DE WINDOWS 98

Un interesante libro que te permitirá tener un control completo de Windows 98, con una información accesible y clara para cientos de opciones de personalización y optimización del Registro. Podrás realizar las modificaciones más complejas del Registro mediante instrucciones paso a paso fáciles de seguir.

Encontrarás rápidamente consejos, atajos y advertencias adicionales que añaden productividad. Localizarás toda la información que necesites, aunque no entiendas la jerga informática o los términos técnicos. El índice está diseñado para ayudarte a encontrar lo que precises, aún no sabiendo claramente lo que buscar.



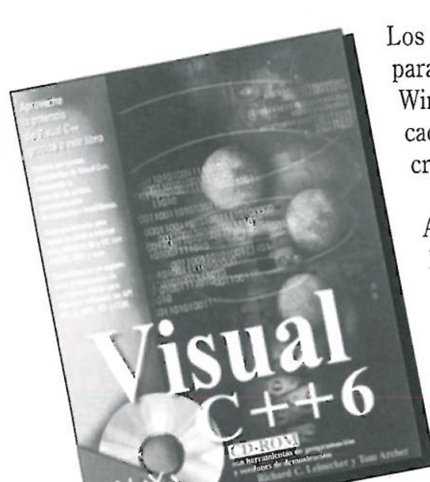
Este volumen te permitirá conocer que tu sistema está optimizado para las aplicaciones específicas con las que trabajas, ya que contiene más opciones de personalización y optimización que cualquier otra obra.

El manual incluye un CD-ROM que contiene Power Tools, una potente herramienta que permite modificar docenas de parámetros del Registro, seleccionando simplemente todas las características que desees mejorar en un sencillo cuadro de diálogo.

Editorial: Prentice Hall
Nº páginas: 499
Nivel: Principiante-intermedio
Autor: Jerry Honeycutt
Idioma: Español
Precio: 6.500 Ptas. (I.V.A. inc.)

VISUAL C++ 6

Gracias a este libro lograrás aprovechar al máximo la potencia de Visual C++. Conseguirás aprender todas las características que necesites para desarrollar proyectos para Windows o también basados en la red a nivel industrial. Podrás conocerlo todo, desde las bases de un menú hasta las entradas a través de un ratón pasando por la programación de scripts ODBC, DAO y ADO.

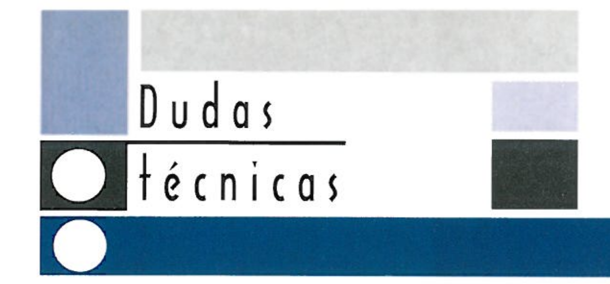


Los útiles consejos que los autores de este manual ofrecen a los lectores, te servirán para encontrar todo lo relacionado con el desarrollo y programación de elementos para Windows, aumentarás las capacidades de las bases de datos ODBC, ampliarás las aplicaciones con la carga dinámica de DLL y librerías de terceros, utilizarás MFC para crear controles ActiveX o ATL para crear pequeños controles para las aplicaciones.

Al mismo tiempo lograrás construir aplicaciones dinámicas para Internet gracias a los trucos, conexiones CHtmlView y DHTML.

El libro va complementado con un CD-ROM en el que podrás hallar numerosas herramientas de programación y versiones de demostración.

Editorial: Anaya Multimedia
Nº páginas: 864
Nivel: Intermedio-avanzado
Autor: R.C.Leinecker y T.Archer
Idioma: Español
Precio: 5.995 Ptas.(I.V.A. inc.)



Dudas técnicas

En esta sección, como cada mes, SÓLO PROGRAMADORES os brinda la oportunidad de encontrar respuesta a las dudas que podáis tener en algún tema relacionado con la programación bajo cualquier entorno de desarrollo.

Pregunta

Hola amigos:

Hace tiempo que leo vuestra revista y, visto el éxito que tiene la sección de preguntas del lector, he decidido animarme a mandar la mía. Antes de nada, quiero agradeceros de antemano vuestra respuesta que, estoy seguro, resolverá todas las dudas que tengo en este tema.

Pues bien, la pregunta es la siguiente. Estoy haciendo un pequeño trabajo de *profiling* para la universidad, en el cual necesito conocer la velocidad de transferencia a través del bus de una tarjeta de vídeo. He realizado un pequeño programa que me permite escribir puntos de color a la tarjeta de vídeo, sin embargo por muy rápido que ejecute el programa, no consigo superar los 2 Mb por segundo. Tengo entendido que las tarjetas actuales dejan transferir información mucho más rápido que eso. Sin ir más lejos, utilizando el juego *Quake*, mi ordenador consigue unos 25 frames por segundo, lo cual ya es mucho más que 2 Mb/s a una resolución de 640x400x256. Si no me equivoco, la fórmula sería de la siguiente manera:

640 x 400 x 1 bpp = 250 Kbytes

250 Kbytes x 25 imágenes/seg = 6.10 Mbytes/segundo

¿Qué estoy haciendo mal? ¿Cómo podría medir la velocidad real, o la máxima de mi tarjeta?

Respuesta

Querido lector:

La velocidad de transferencia de una tarjeta de vídeo, o de cualquier periférico, a través de un bus, depende de muchos factores, por lo que no se puede hablar de una única velocidad de acceso a la memoria de la tarjeta. Entre los diversos factores que toman un papel decisivo en la velocidad de la tarjeta, podemos citar los siguientes:

- El tipo de *bus* utilizado marca claramente el comportamiento de una tarjeta de vídeo. Si comparamos dos tarjetas de idénticos *chipset* y diseño, una de las cuales utiliza *PCI* mientras que la otra utiliza *ISA*, podremos observar que la primera es mucho más rápida que la segunda.
- El alineamiento de los datos juega también un papel clave a la hora de conseguir rendimiento. La memoria del *PC*, así como la de la tarjeta de vídeo, suele estar organizada en palabras de 32 *bits*. Si escribimos datos no alineados con las fronteras de estas palabras, tendremos que acceder a varios bancos para un único punto o bien varias veces a la misma palabra para colocar dos puntos. El resultado final es una pérdida de velocidad bastante apreciable.
- Una *CPU* lenta o muy cargada puede no ser capaz de enviar los datos a toda la velocidad que acepta la tarjeta. Si lo que estamos midiendo es una velocidad máxima de acceso, deberemos asegurarnos de que la *CPU* está prácticamente desocupada.
- El modo de vídeo que utilicemos juega un papel fundamental. El diseño de una tarjeta de vídeo moderna fuerza al ingeniero a aceptar penalizaciones a cambio de conseguir mejores rendimientos. Debido a ello, los modos de texto y 16 colores son infinitamente más lentos que los de 256 y 16 *bits*. Además, si utilizas 16.7 millones de colores, puedes utilizar dos tipos de arquitectura: 24 y 32 *bits*. La primera transfiere menos bits, sin embargo su acceso no está alineado como en el segundo caso.

- El método de acceso es también muy importante en cuanto tenemos modos de vídeo con más de 64 Kb de memoria.
- La cantidad de datos transferida en un solo bloque condiciona de una manera muy clara el rendimiento del acceso. Todos los *buses* suelen tener una latencia inicial, debido a la cual es más rápido transmitir muchos datos de una vez que enviar la misma cantidad de información en varios bloques.

Una vez hemos decidido el tipo de programa que vamos a crear, es el momento de elegir el lenguaje en el que programarlo. Tenemos tres opciones:

1. Un lenguaje de alto nivel como pudiera ser *C++*. Es la forma más sencilla, ya que además de una sintaxis cómoda y ágil, existen una gran cantidad de librerías gráficas a nuestra disposición para las engorrosas tareas de selección de modo, borrado de pantalla, dibujo de puntos, etc...
2. Un lenguaje de nivel intermedio, como pudiera ser el *C*, el cual nos permite utilizar punteros a memoria de una manera más directa y sigue proporcionándonos librerías que facilitan enormemente la creación del programa, aunque la sintaxis empieza a requerir un conocimiento más profundo de la arquitectura del sistema.
3. La solución de más bajo nivel, la cual nos permite un control absoluto sobre nuestro sistema es, sin duda, un lenguaje *Ensamblador*. Sin embargo, si elegimos esta opción, tendremos que generar todas las líneas de programa necesarias para activar el modo de vídeo, etc.., las cuales son de una complejidad bastante alta.

Desgraciadamente, los lenguajes compilados no nos proporcionan un control absoluto sobre el comportamiento de los datos en su viaje a la tarjeta de vídeo. Si a esto sumamos que los compiladores no son omniscientes a pesar de todas las optimizaciones incorporadas, vamos a encontrar que el lenguaje válido para construir un banco de pruebas de estas características es el *Ensamblador*. Sin embargo, las tareas de preparar el modo de vídeo y restaurar el modo de texto son demasiado complejas y engorrosas como para codificarlas en ensamblador. Por lo tanto, utilizaremos la aproximación preferida de los ingenieros: una solución de compromiso.

Construimos nuestro programa en *C* para aprovechar las funciones de librería que nos proporciona el sistema. Sin embargo, el banco de pruebas lo vamos a programar en *Ensamblador*, utilizando la capacidad que tienen los compiladores actuales de permitirnos incluir este lenguaje entremezclado con código *C*. A continuación exponemos, en pseudocódigo, el programa final:

```
/* Inclusión de las librerías
    necesarias */
#include .....
int  main ( argc , argv )
    int argc;  char * argv[];
{
/* declaración de variables */
.....
/* Inicialización de variables */
.....
/* Preparación del modo de vídeo
    utilizando las funciones del
    sistema */
/* Obtención de la posición de
    memoria donde se encuentra el
    vídeo */
....
/* Puesta a cero del cronómetro */
/* Llamada a  la función ensam-
    blador */
mymemset ( direccion , ancho,
    alto );
/* Verificación del valor del
    cronómetro */
/* Vuelta al modo texto e impre-
    sión de resultados. */
}
```

A continuación describimos la función en lenguaje ensamblador:

```
static inline void
    *__mymemset2(void *s, size_t
width, size_t height)
{
   __asm__(
  "cld\n\t"
 "1:\n\t"
  "movl %%ecx,%%eax\n\t"
  "movl %%ebx,%%edi\n\t"
  "2:\n\t"
  "movw %%eax,(%%edi)\n\t"
  "addl $4,%%edi\n\t"
  "decl %%eax\n\t"
  "jne 2b\n\t"
  "addl $1024,%%ebx\n\t"    /*
mode 320 width, unbanked */
  "decl %%edx\n\t"
  "jne 1b\n\t"
  ::"b" (s),"d" (height), "c"
(width)
  :"ax","bx","cx","dx","di"
   );
return s;
}
```

Pregunta

Me dirijo a Sólo Programadores, para obtener respuesta a una duda que se me ha presentado a la hora de programar un acelerador gráfico. Estoy haciendo un juego 3D sin utilizar *Windows* para así conseguir obtener toda la velocidad posible del ordenador. El único problema es que me tengo que programar yo mismo las rutinas gráficas de la tarjeta de vídeo. He probado con varias y, aunque las librerías *VESA* que tengo me permiten acceder a los distintos modos y resoluciones de las tarjetas, encuentro que tengo algunos problemas de velocidad a la hora de presentar los gráficos. Me bajé de *Internet* el manual de programación de una tarjeta y parece que lo que necesito es una cosa que se llama expansión de color. Sin embargo, por más que hago pruebas y le doy vueltas, no encuentro el modo de utilizarla. ¿En qué consiste la consabida expansión de color? ¿cómo puedo utilizarla? ¿tienen esto todas las tarjetas de vídeo? En fin, como veis me encuentro un poco perdido. Si podéis ayudarme os lo agradezco de antemano.

Respuesta

Para responder a tus preguntas, tenemos que hacer un poco de historia de la arquitectura de las tarjetas gráficas. Originalmente, y tras varios tipos de adaptadores gráficos de bajas resoluciones y poca capacidad de color, *IBM* propuso el estándar *VGA* que definía resoluciones de hasta 640x480 con 16 colores en pantalla y de 320x200 con 256 colores. Después los fabricantes ampliaron los modos y resoluciones disponibles, hasta el día de hoy en el que podemos encontrar tarjetas capaces de presentar 1600x1280 puntos en pantalla con cantidades de colores casi infinitas.

Uno de los métodos clásicos de aceleración es lo que se denomina expansión de color. Este sistema se basa en el hecho de que, en un gran número de ocasiones, sólo utilizamos dos colores para dibujar: fondo y primer plano. Para ello, preparamos dos registros en la tarjeta:

- **Primer plano:** en éste almacenamos el código de color que corresponde al color de primer plano.
- **Fondo:** aquí introducimos el color correspondiente al fondo.

De esta manera, nosotros ya no enviamos 100 bytes a través del bus (modo de 256 colores), sino que mandamos una máscara de dibujo, en la que los 1's se corresponden con el color de primer plano y los 0's con de fondo. De esta manera, al mandar un único byte, el acelerador va a pintar 8 puntos, expandiendo los 1's y los 0's con el contenido de color de los registros. Gracias a este sistema, reducimos a 1/8 el tráfico en el *bus* para modos de 256 colores, siendo las ganancias aun más acusadas en modos de 24 y 32 bpp. Así, en lugar de enviar 100 puntos, sólo hace falta mandar 13.

En cuanto a tu última pregunta, la respuesta es sí. La gran mayoría de las tarjetas de vídeo ofrecen algún tipo de expansión de color como parte de la aceleración 2D. Sin embargo, hemos de decirte que cada tarjeta tiene su propio método, así como una manera única de utilizarlo.

Pregunta

Soy un programador que, ilusionado por el código fuente que incluís en la revista, sobre el *FreeAmp*, lo abro con el *Visual C++* v 6.0 y, cuál no será mi sorpresa cuando compruebo que tiene varios errores y no es posible compilarlo y linkarlo correctamente. Algunos errores serían fáciles de solucionar, como por ejemplo alguna que otra declaración de miembros de una clase hecha de una forma e implementada de otra diferente, pero el grueso del problema no consigo solucionarlo ya que el proyecto es bastante amplio. Me podríais indicar qué es lo que se puede hacer para que funcione correctamente. Gracias.

Respuesta

Estimado lector:

Hemos comprobado que, efectivamente, la versión a la que te refieres en tu consulta consta de numerosos errores que hacen que su compilación no sea una tarea fácil y rápida como es de esperar, por lo que es muy recomendable utilizar la última versión ya disponible, 1.2.3, tanto en los entornos *Windows* como *Linux*, ya que contiene gran funcionalidad.

Los pasos para realizar la compilación de *FreeAmp v1.2.3* son bastante sencillos, aunque al utilizar *Visual C++* se debe tener cierta precaución al intentar compilar un tipo de proyecto. En el caso de la compilación usando *Visual C++* bajo *Windows*, el proyecto principal se localiza en el directorio de instalación de *Freeamp* y a continuación en los directorios *base\ win32\prj*. El fichero **freeamp.dws** organiza todos los subproyectos y construye todos los módulos. En este punto hay que decidir que tipo de configuración de proyecto se va a utilizar. Se dispone de las siguientes:

- *Win32 Debug*
- *Win32 Release*
- *Win32 NASM Debug*
- *Win32 NASM Release*

Inicialmente la opción más simple de utilizar y más recomendada es la *Win32 Release*. Para ello, en la barra de menú de *Visual C++* seleccionamos la opción *Build*, y dentro de la misma *Set Active Project Configuration*. Seleccionamos la entrada *freeamp-Win32 Release* y pulsamos el botón de aceptar. Para crear el ejecutable **freeamp.exe** y los ficheros requeridos para su funcionamiento, se vuelve a seleccionar la opción *Build* y dentro de ésta la opción *Build freeamp.exe F7*. Tras todo el proceso de compilación y linkado y si todo ha ido bien, como debería, se obtiene el mensaje :

```
---Configuration: freeamp - Win32
     Release---
freeamp.exe - 0 error(s),0 war-
     ning(s)
```

por lo que ya se dispone del reproductor de MP3. Para su perfecta utilización no hay mas que crear un directorio que contenga el archivo **freeamp.exe** y un subdirectorio que tiene que llamarse **plugins** que contiene estos archivos :

- fileinput.pmi
- freeamp.ui
- httpinput.pmi
- obsinput.pmi
- rainplay.ui
- simple.ui
- soundcard.pmo
- xing.lmc

que se encuentran en el directorio *base\win32\prj\plugins.*

Una vez realizados estos pasos, podrás disfrutar de *Freeamp* para reproducir tus ficheros tipo *MP3* favoritos.

La llave para acceder a las nuevas tecnologías



Feria Internacional de Informática, Multimedia y Comunicaciones

Madrid, 2-7 Noviembre de 1999

Parque Ferial Juan Carlos I
28042 Madrid
Apdo. de Correos 67.067 • 28080 Madrid
Tel.: 91 722 53 30 / 50 00 • Fax: 91 722 58 07
e-mail: simo@ifema.es • www.simo.ifema.es



